DATA BASE THEORY AND DESIGN

NORSK DATA A.S

DATA BASE THEORY AND DESIGN

Preliminary Issue

This manual was originally produced in Norwegian under a project in SIAG regi, called Data base metodikk. It has been translated to English and a new first chapter added by A/S NORSK DATA-ELEKTRONIKK.

Jeremy Salter
18/10/1976

REVISION RECORD

Revision	Notes
12/7	Original Printing

DATA BASE THEORY AND DESIGN
Publ. No. ND-60.078.01
December 1978

NORSK DATA A/S
Lørenveien 57, Postboks 163 Økern,
Oslo 5, Norway

DATA BASE THEORY AND DESIGN

Contents

1 INTRODUCTION TO DATABASES AND DATABASE CONSTRUCTION

1.1 Information, data redundance and data dependance

Any organisation, however large or small needs information in order to be able to function. This may be information related to the market to whose satisfaction the organisations activity is directed, or information related to the internal functioning of the organisation. In a small firm it may be possible for one, or a small number of persons to have a complete awareness of all these information needs, but in large organisations one finds seperate departments each with its own information requirements.

We can regard information as data looked upon in a certain way, and this implies that the same data can provide different information depending upon who is using it, or upon how it is used. For example the production planning department of a firm is interested in information regarding new orders so that they can be completed inside given delivery dates. Those responsible for stock holdings of parts or rawmaterials are interested in the same orders because they must ensure that production will not be delayed by lack of material or parts. For both departments the data from which they receive the necessary information will oginate from a sales or marketing department. It may seem pretty obvious that these various departments should be operating with the same data, but this may not be the case for several reasons. If each department has its own information system, then the data communicated to it will have to be fed into its own system. The data may be lost, delayed or altered as a result of this communication, or if it is a new type of data it may even be (temporarily) unusable from another information sub-systems point of view.

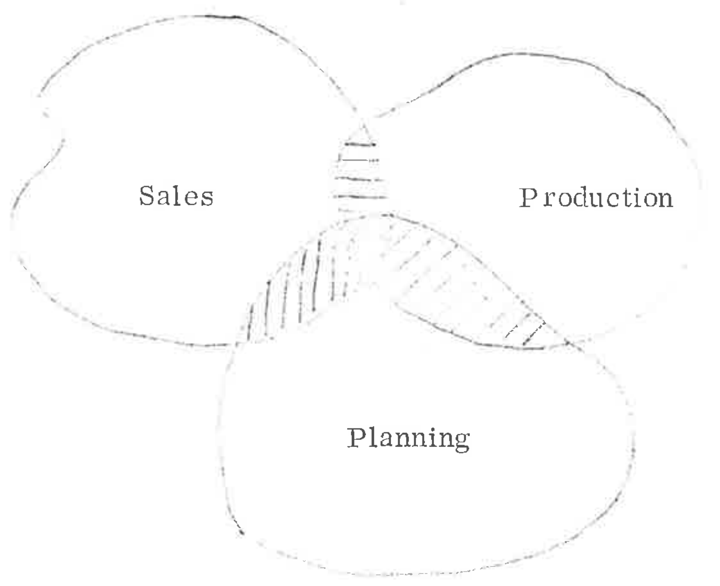
This is illustrated in figure 1, where each circle represents the data needed or used in connection with an organisations sales, planning, and production functions. For a small organisation only one source of data exists, and the data necessary to provide information for more than one function (overlapping of the circles) is always the same data independently of how it is used (fig. 1A). But when an organisation expands, the volume of data required by each department expands and tends to "seperate" (fig 1B) leading to data redundance (more than one copy of the same data, different values for the same data).

Above we have discussed and illustrated the problems in connection with the maintenance of data, the other side of the matter arises in connection with the use of data.

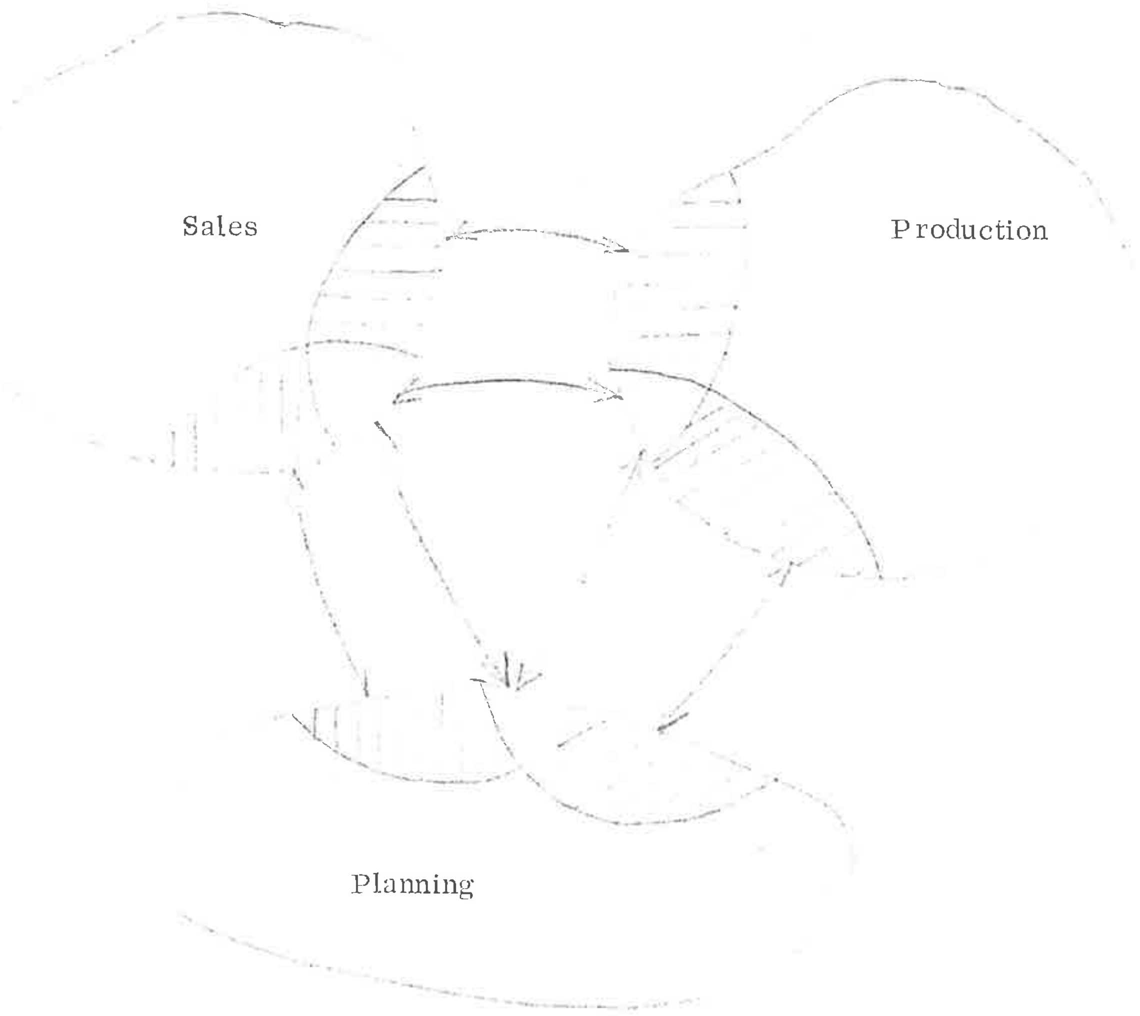
This is more easily discussed in relation to comp$\overset{u}{\wedge}$terised information systems. Each department in an organization has its own data pool which may be in the form of magnetic tape files, or direct access files. The records on these files will consist of different data organized in a way which is only relevant for the one department. Information will be obtained by running various programmes which use some or all of the data in various ways and these programmes will be dependent upon being aware of the physical layout of the records they process. Increasing the data content of records, or changing the way in which it is arranged will mean that some or all of the computer programmes using the data will have to be rewritten. This so called "data dependance" makes the changing of existing traditional systems both costly and time consuming and the development of an organisation itself, and its ability to adapt and react to changing external events will be limited correspondingly depending upon how dependent it is upon the use of the information system. This is illustrated in fig.2.

The solution to these problems, the avaidance of data redundancy and data dependence and their consequences is currently called the "database approach" (see fig. 3)

Fig. 1

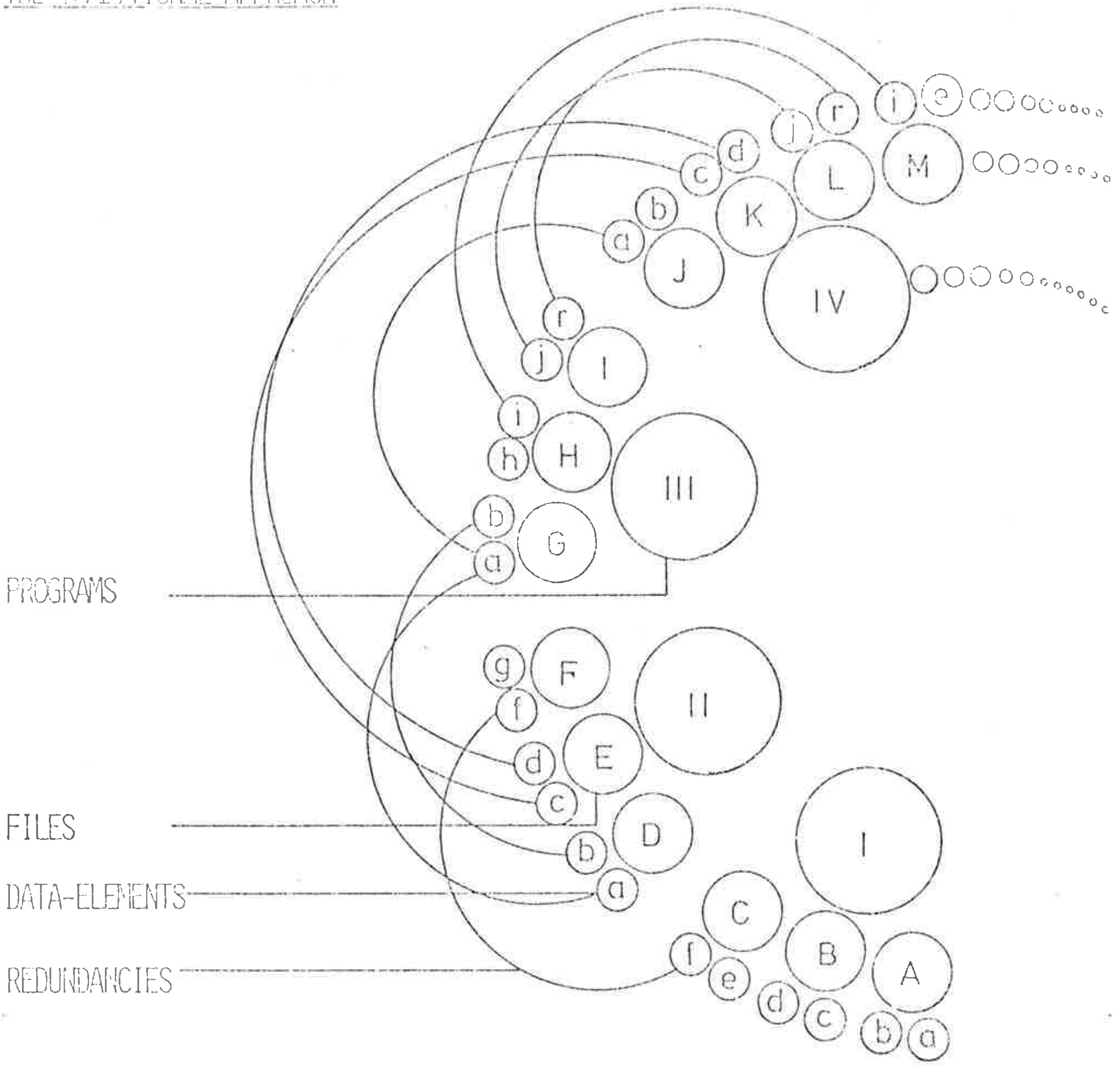


Common data requirements (shaded or lined), "pooled" in a small organisation, small data volumes, no data redundancy.



Common data requirements by communication (arrows) and redundance (shaded or lined), large data volumes.

THE TRADITIONAL APPROACH

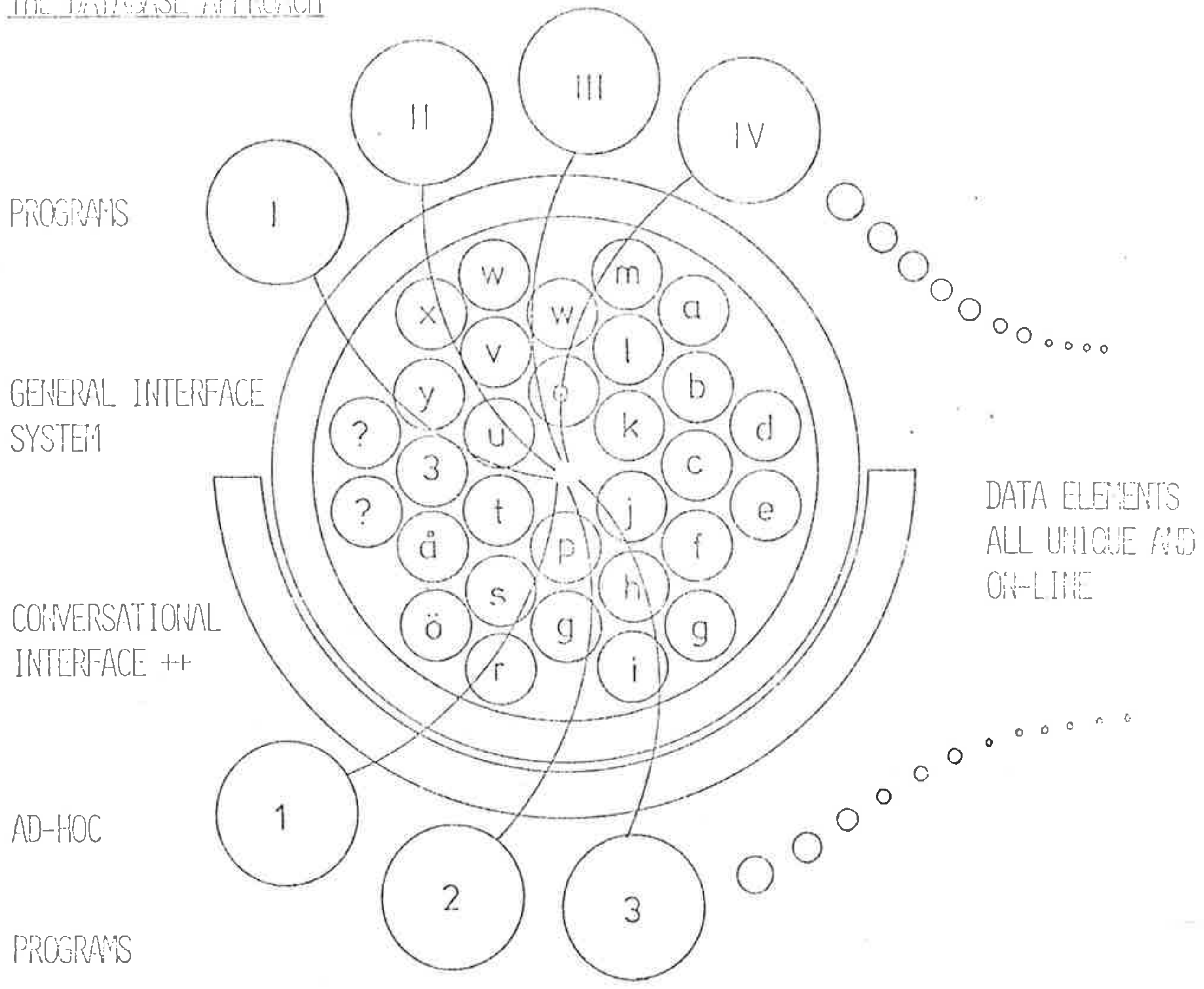


- DATA REDUNDANCE
- DATA DEPENDENCE

GIVING RISE TO

- RESOURCE PROBLEMS
- MAINTENANCE PROBLEMS (UPDATING)
- RIGIDITY

Fig 2.



- REDUNDANCY AND UPDATE PROBLEMS AVOIDED.

- PROGRAMS NEEDING THE SAME INFORMATION USE THE SAME VERSION OF THE DATA.

- DEVELOPMENT OF NEW PROGRAMS IS EASIER, QUICKER AND CHEAPER.

- AD-HOC QUESTIONS CAN BE ANSWERED.

- USER PROGRAMS INDEPENDENT OF HOW DATA IS STORED PHYSICALLY.

Fig 3.

All the data required by an organisation is put into a common pool, the database. The responsibility of maintaining and protecting the data is to a large extent removed from the various user departments to a database administrator function. This solves the problem of data redundance.

The problem of data dependency is solved, partialy anyway, by insisting that all access to the database go via standard database software, and that all reference to data, record types or specific data elements is symbolic, or at the logical level. To give a simple example, each data element in a person record will have a name, like AGE, SURNAME, ADDRESS and it is sufficient to give these names is one wishes to obtain a persons age, surname and address from his person record. One does not have to know anything about where the data elements are placed in relation to the start of the record, or where the record physically is stored. The introduction of new data elements into records will not affect existing user programs so long as they are not to make use of any of these new data elements.

Another advantage of the database approach is that it is always possible to write new application programs to use data already in the database to provide a new type of information.

And to the extent that a general enquiry language is supported by the database system one is using, it should be possible to sit on line at a terminal and access data in the database.

There is one important point in connection with the conversion to or introduction of database philosophy into an organisation. How does one go from the situation illustrated in fig. 2, or from a situation involving no use of computerised data handling techniques, to that illustrated in fig. 3 ? The number of data elements may be very large and they may be combined and used many different ways, and often a single person will not be able to visualize all these at one time. It is therefore very important to have some kind of method or procedure which allows one to analyze required information needs in terms of the data to be stored in the database and of the ways in which it can be used. The remaining chapters in this manual are devoted to describing such a (semi-formal) method.

1.2 Database Systems and File Systems

For those who are accustomed to traditional systems where application programs work directly upon records held in files, it is improtant to have the difference and the relation between a database systom and a file system clear. This is illustrated in fig. 4 where we see that using a database system removes the user one stage further from the physical storage of data by insisting that he use a standard interface, the database software, lying between him and the data he wishes to use.

1.3 Database Definitions

A number of database definitions are given below:

1. A collection of shared data
2. The data necessary for the running and control of an organisation
3. All record occurrences, set occurrences and areas controlled by a given schema.
4. A collection of structured data reflecting the working of an organisation and serving its requirements for storing and supplying data
5. A collection of structured data belonging to an organisation, and serving its information requirements, which can be simultaneously accessed by a number of users.

FILE SYSTEM - DATABASE SYSTEM

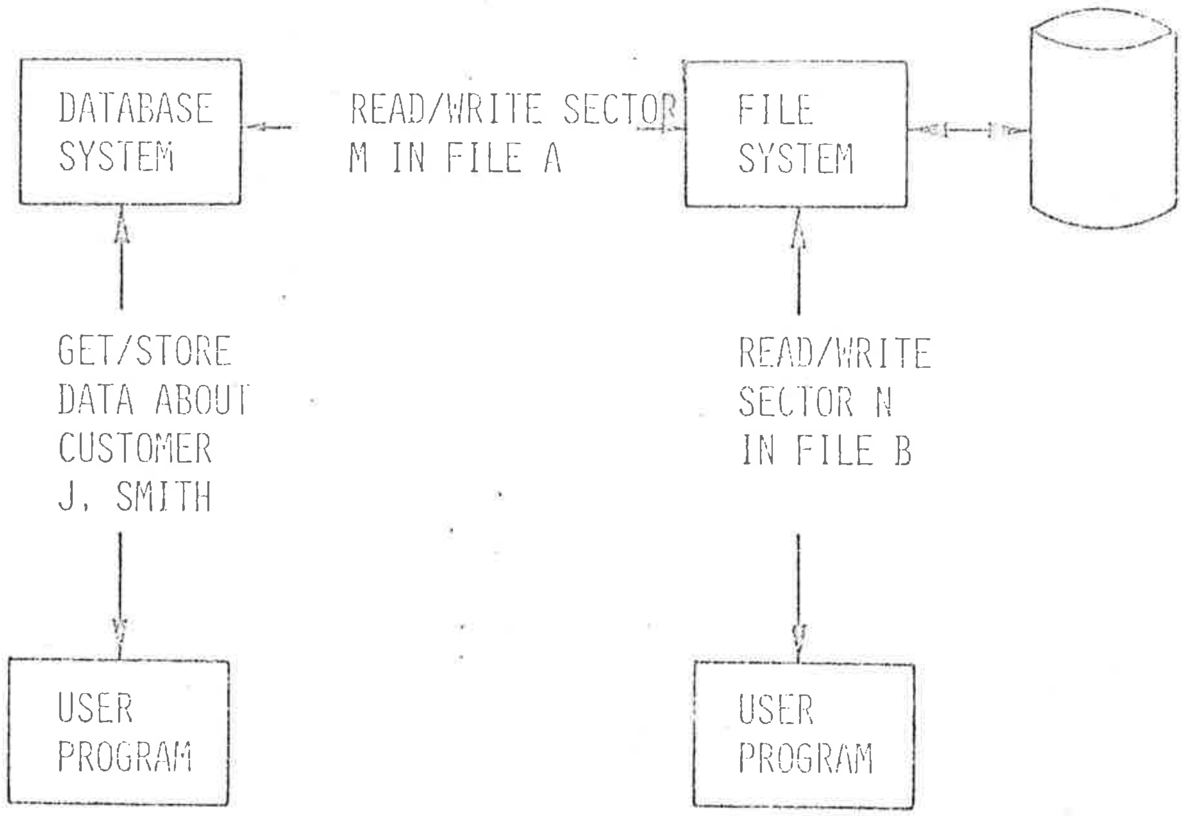

DATABASE SYSTEM
READ/WRITE SECTOR M IN FILE A
FILE SYSTEM
GET/STORE DATA ABOUT CUSTOMER J. SMITH
READ/WRITE SECTOR N IN FILE B
USER PROGRAM
USER PROGRAM

Fig. 4.

1.4 Database System Functions

The services or functions which one should expect from a database system are listed below:

1. Control over the duplication of data to avoid unnecessary data redundance.

2. To provide storage, structuring and access mechanisms so that applications can be run as efficiently as possible.

3. To provide for the controlled simultaneous access of data by more than one application.

4. To provide for the seperation of application programming logic from any knowledge of physical data structures.

5. To provide reorganisation facilities for existing data, and the possibility of removing old or adding new data types so that the database can be adapted to changes in the organisation which it serves.

6. To provide privacy controlls to prevent unauthorised access to data.

7. To provide for the integrity of the data in the database and avoid its loss or damage. To give good back up facilities and recovery facilities in the event of break down.

8. To provide for the retrieval, storage, and deletion of data in an efficient and measurable way.

9. To support the use of the database system via a number of highlevel languages.

1.3 Construction of Database

The remainder of this chapter describes the approach described in this manual, namely how a database can be used as a model of reality, of for example an organisation. The database is regarded as a collection of technical data representing events, states, points in time, physical objects and so on which are used to produce and handle information related to the reality which the database is meant to represent. One difficulty with using the database as a model is that one must understand the data technical aspects of the model. Another problem is that of distinguishing between the data technical aspects and the conceptual aspects during the construction of the model.

To take account of three difficulties we have here chosen to construct the database in two clearly separated phases. In the first phase we describe reality. This is done here using a conceptual model which is a logical description independent of the data technical aspects. The process of going from the conceptual model to its implementaion is described using an implementation structure.

This two stage description process is shown in figure 5.

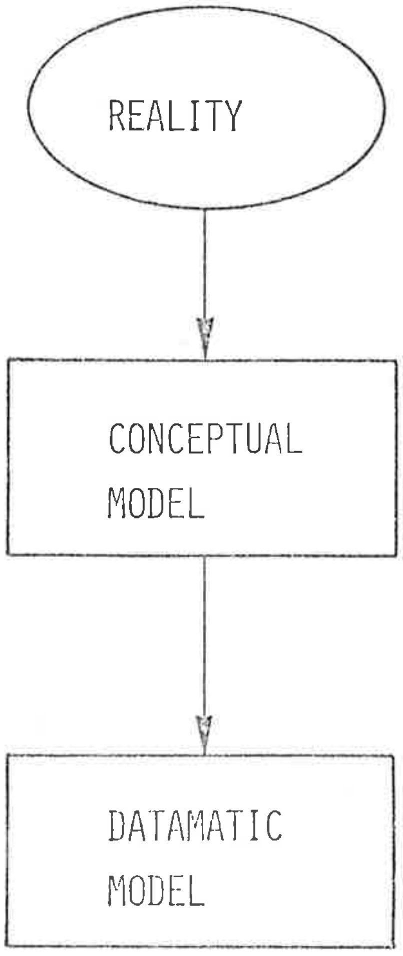


Fig. 5

A closer definition of the conceptual model and implementation structure is given in chapter 2.

Reality can be regarded as consisting of a number of "solid" elements, called objects, together with a number of relations between these objects. Objects can be a customer, a product and so on. Relations are the various interdependencies which exist between the objects.

For example, an order can be regarded as a relation between a customer and a product.

Objects and relations exist independently of databases and information systems. The aim in constructing a database is to provide a data oriented description of the objects and relations of reality, which enables one to obtain information from this description (or model) using simple information seeking procedures.

For example, if one has a database in which information about customers, products and orders is stored, then it should be possible using simple procedures to obtain information about which orders a customer has, which customers have orders for a given product, and so on.

When the conceptual model is to be implemented one needs details of how often the information describing each object or relation is to be used. These details are found by analysing the way in which the projected database is to be used. It is also necessary to know something about the characteristics of the database system which is to be used. The implementation of the conceptual model is described in terms of the implementation structure in which the access mechanisms, storage mechanisms and layout of the various record types are described.

As a basis for the use of the construction rules on a projected system, one should have arrived at a description of the systems information requirements preferably via an information analysis procedure. Thus the description should be arrived at via some formal procesure like the ISAC method, or one should have carried out a general analysis of the "reality" which is to be described using the database, together with an analysis of the information requirements which the database is to satisfy. One should also have a working knowledge of the most common structure mechanisms used in modern database systems in order to apply the structuring rules effectivly.

The aim here is to make the construction of the conceptual model independent of the database system under which it is eventually to be implemented. The conceptual model can, at any rate in principal be implemented without the use of a database system.

With the transformation to the implementation structure the physical conditions under which the database is to be implemented (the actual database system, the computer, the storage media etc.) together with details of how the database is to be used must be considered.

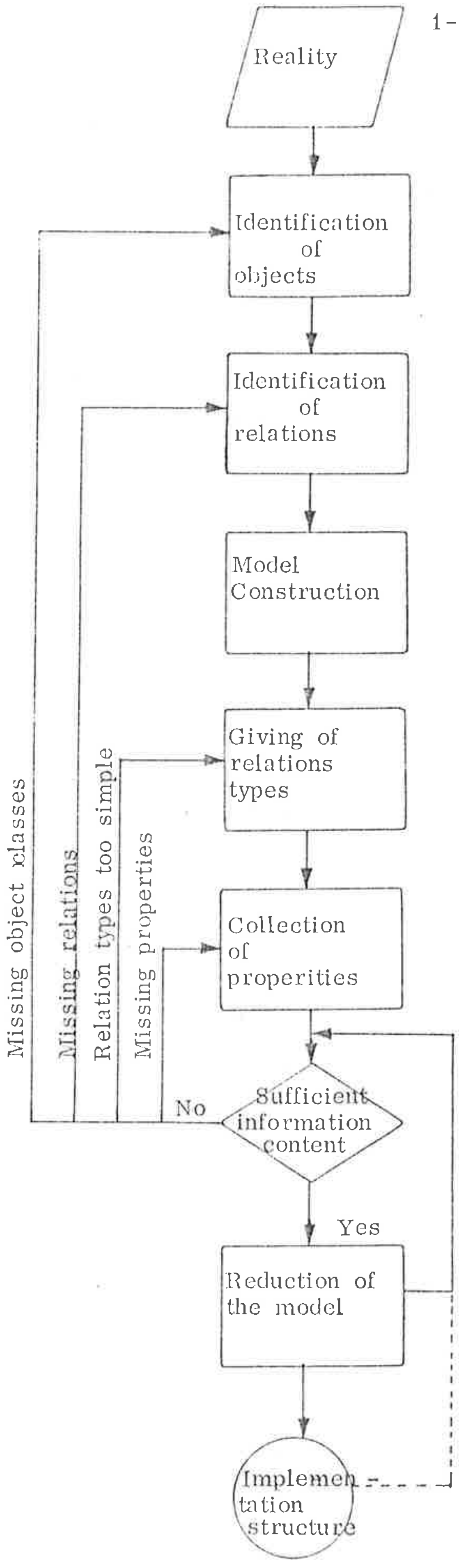
One can in cases where the physical conditions demand it, go back and change the conceptual model. The various stages of the construction process and the order in which they are to be carried out are shown in figure 6A and 6B.

The end result of such a construction process is a "high level" documentation of the record types and the structure of which the database is composed. This documentation creates a basis for a detailed structure documentation. In addition one should have expressed ones need for information and how this is satisfied by the databases content and structure. This then provides a high level specification of the application programs which are to use the database.

The most important results of applying the set of rules and notation to be described are as follows :

- the various parts of the construction process are forced into a definate order
- it is easier to picture the total system
- one remains independent of the actual database system to be used for as long as possible.

The rules are designed in such a way that one works with the logical structure until one has a model which covers the required need for information in the simplest possible way. Logical simplifications can be made independently of details related to the physical implementation.



Reality
Identification of objects
Identification of relations
Model Construction
Giving of relations types
Collection of properities
Sufficient information content
No
Yes
Reduction of the model
Implementation structure
Missing object classes
Missing relations
Relation types too simple
Missing properties

Input to the construction process is ones view of reality described in forms of objects and activites.

Objects are identified by a name and their identifying properties.

Relations are identified by a short description and any relevant identifying properties.

The model is then drawn up using the given symbols

The type of each relation is drawn in on the model.

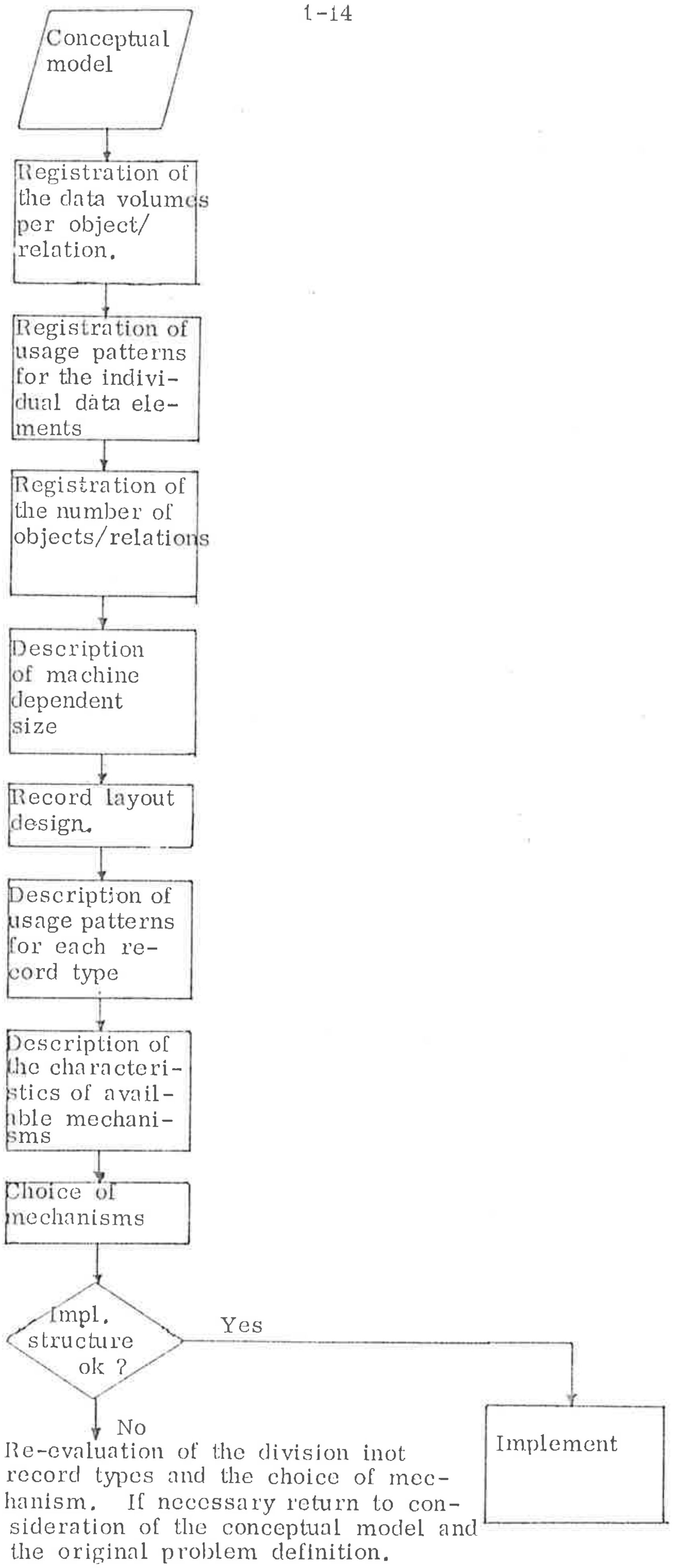
Remaining properties (attributes) collected for both objects and relations.

Relevant questions are put to the model to test whether it has sufficient information content.

The model is reduced mechanically and its information content is tested after each reduction.

When all reductions have been made the design of the implementation structure can begin.

Fig. 6A. Construction of the Conceptual Model.



Conceptual model
Registration of the data volumes per object/ relation.
Registration of usage patterns for the individual data elements
Registration of the number of objects/relations
Description of machine dependent size
Record layout design.
Description of usage patterns for each record type
Description of the characteristics of available mechanisms
Choice of mechanisms
Impl. structure ok ?
Yes
Implement
No
Re-evaluation of the division inot record types and the choice of mechanism. If necessary return to consideration of the conceptual model and the original problem definition.

Fig. 6B. Designing the Implementation Structure.

2 NOTATION AND CONCEPTS

This chapter introduces the concepts and symbols which are used to describe the conceptual model and the implementation structure.

2.1 The Conceptual Model

To describe reality, we use the following concepts which, although familiar, will be defined: objects, relations, properties and identification.

2.1.1 Objects

Physical objects, such as, people, machines, products, etc. Objects have properties which can be measured or observed in some way. For example, colour, shape, weight, etc. Objects can always be identified by their properties as long as these are of sufficient number and detail. In many cases, an identification system for objects already exists, for example, the names of people, their personal identification number, product number, especially where the need for such identification exists. A person name is not generally unique, but personal identification numbers can be unique within one country, and product numbers can be unique in one factory. Such identification numbers can be regarded as artificial, but become real properties in certain cases, as, for example, if one writes the product number on the product. Objects can be classified via their properties, all red products for example.

In addition to physical objects, we will also operate with what can be called abstract objects. A number of relations (see the definition below) have often names and an identification system, as with order and order number, job with job number. It may often be advantageous or suitable to treat a relation as an object, but it is difficult to give exact rules as to when this should be done. A general rule is that all relations which normally have their own identifications system, can be regarded as objects.

Relations between objects in the model and objects lying outside the model will also often be treated as objects.

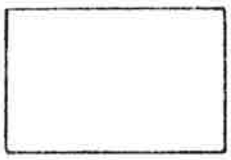
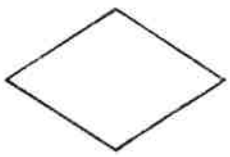
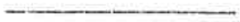
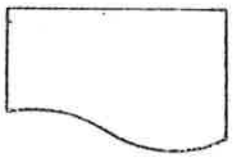
One should always look carefully at the correspondence between the conceptual model of reality and reality itself to ensure that this is unambiguous. For example, two orders with different order numbers but with identical properties, such as, customer, product, and date, may exist. This situation could arise if the same order, that is, agreement with a customer, had, in error, been recorded twice, or it could be that two identical orders had been received and recorded on the same day. In such cases, one could avoid ambiguity by including the time of day with the date.

An object is said to belong to one object class. All objects in an object class are of the same type and the object class is given a name, for example, person.

2.1.2 Relations

Relations describe the connections between objects. For example, a relation between two people could be that they are married, an order can be regarded as a relation between a customer and a product, a job can be regarded as a relation between a machine, a person and a product. Relations have properties which describe which objects are related by the relation, when it existed, how long it lasted and so on. A relation can also be identified by using its properties, provided these are described in sufficient detail and a sufficient number are considered. As mentioned above, one often introduces an identification system for relations, such as, job number or order number. These can, as for objects, be called artificial properties, but they cannot be made real by writing the identification upon the relation since that has no physical existence. For example, a job number cannot be written upon a job, but it can be found upon the job sheet which itself is a description of the job. The job number will then identify the job sheet unambiguously, but the job sheet will not identify the job any further than the description to be found upon the job sheet. Relations can also be defined using the objects with which they relate. For example, if an order is regarded as a relation between a customer, a salesman and a product, then the identification of these three classes can be made the identifying property of the relation.

2.1.3 Symbols and Concepts for the Conceptual Model.

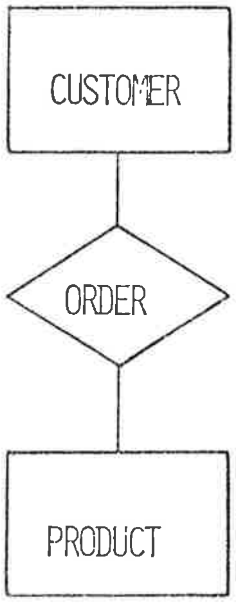
	OBJECTS E.G. PEOPLE, PRODUCTS, BUILDINGS. CAN BE UNIQUELY IDENTIFIED USING ONE PROPERTY
	RELATIONS BETWEEN OBJECTS. E.G. ''HAS ORDERED'' IS A RELATION BETWEEN A CUSTOMER AND A PRODUCT. IDENTIFIED BY THE OBJECTS IT IS A RELATION BETWEEN.
PROPERTIES OR ATTRIBUTES	BELONG TO OBJECTS OR RELATIONS AND DESCRIBE THEM AS FULLY AS IS REQUIRED.
	CONNECTION BETWEEN A RELATION AND THE OBJECTS IT RELATES.
	A TRANSACTION REPRESENTING A DEFINITE NEED FOR INFORMATION FROM THE MODEL.

2.1.4 An Example

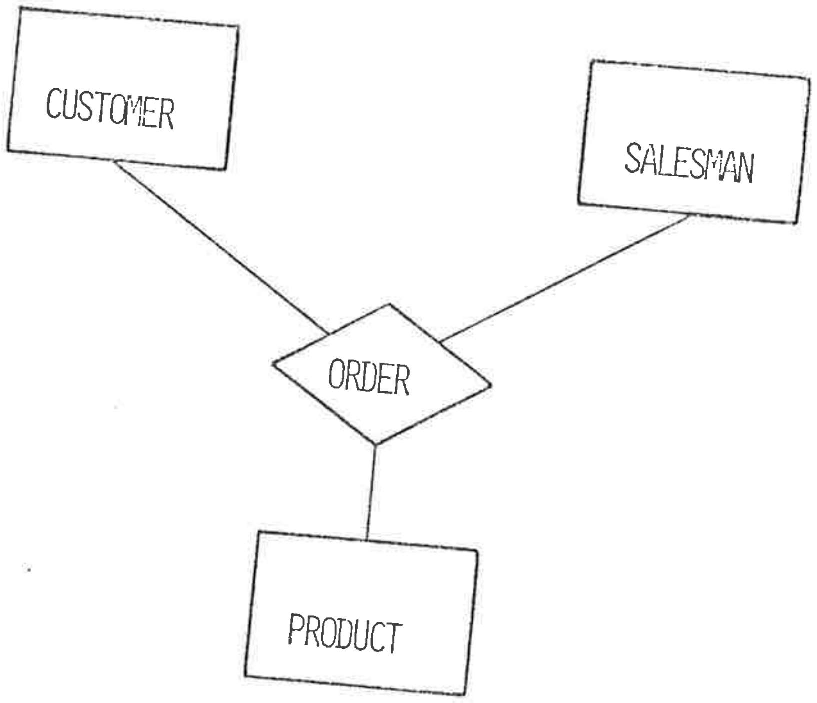
A simple example will be given to illustrate the use of the symbols.

We will make a conceptual model of the situation where customers give orders for products. Customers and products will be regarded as objects, and orders as a relation between customer and product. We assume that one numbering system exists for customers, another for products, and a third for orders.

The conceptual model will then appear as illustrated below.



If we wish to include in the conceptual model that salesmen receive orders, then the object class order can be regarded as a relation between customers, salesman, and products, and the model will then be:



2.2 The Implementation Structure

In order to describe the data structure which is to represent our conceptual model in a computer, we use the following concepts: record, set and access key.

2.2.1 Record

A record consists of a number of data elements and can describe an object or a relation. Each data element describes some property of the object or relation.

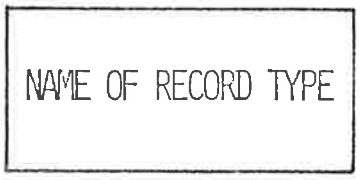
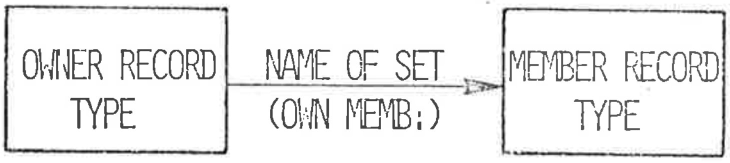
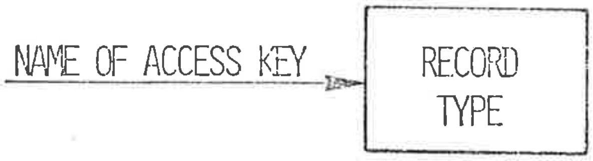
2.2.2 Set

Records can be coupled together in sets where one record is owner and one or more records are members of the set. A set can be used to describe a relation to which one does not wish to attach any data or it can represent a classification of objects or relations (implemented as records).

2.2.3 Access Keys

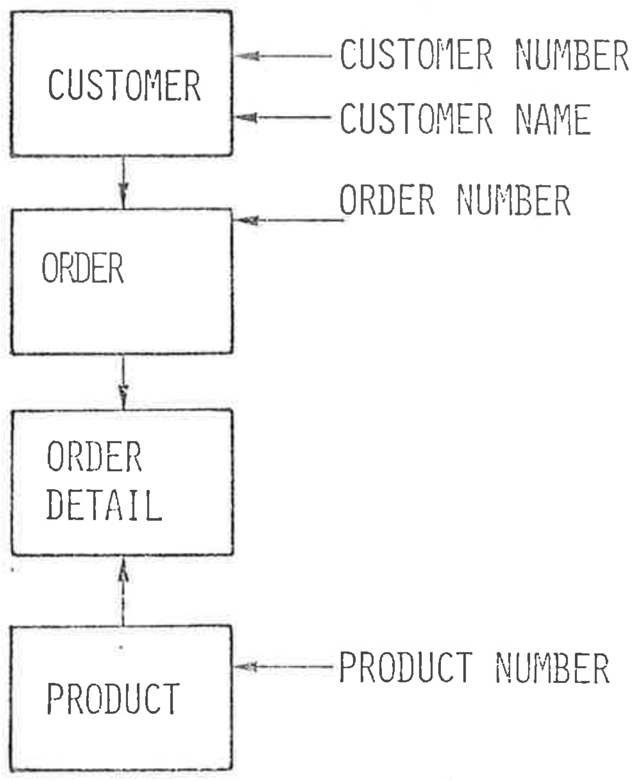
An access key is a property or data element which is to be used to find a record. An access key can be unique, as is person, identification number, or ambiguous, as in the case of a person name.

2.2.4 Symbols and Concepts for the Implementation Structure

TERM	SYMBOL
RECORD TYPE	
SET	 OWNER RECORD TYPE NAME OF SET (OWN MEMB.) MEMBER RECORD TYPE
ACCESS KEY	

2.1.5 An Example

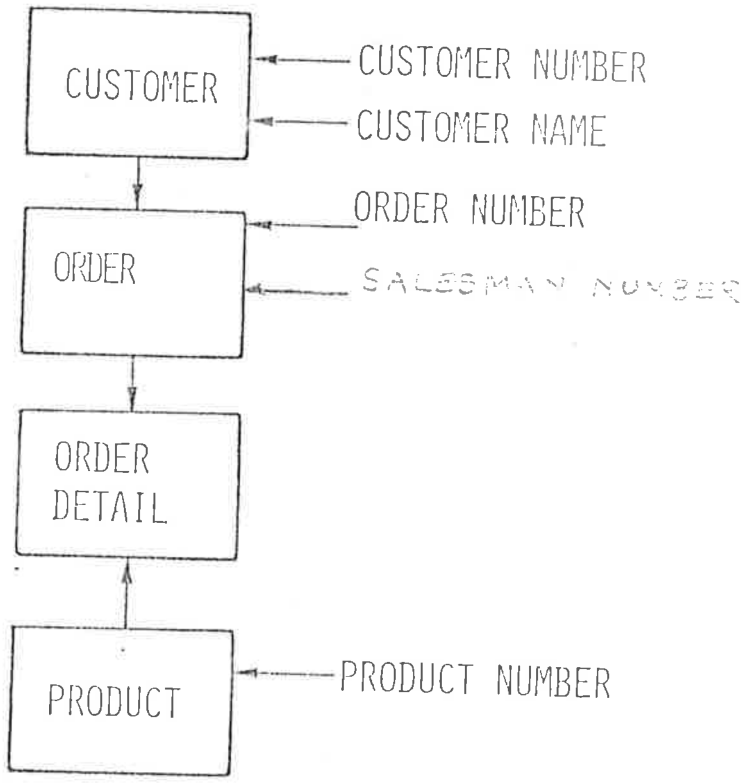
Taking the same example, used for illustrating how the conceptual model is applied, we indicate how the above symbols are used. We assume that each order consists of a variable number of order details, and that each order detail refers to one product only. The following implementation structure can then be made.



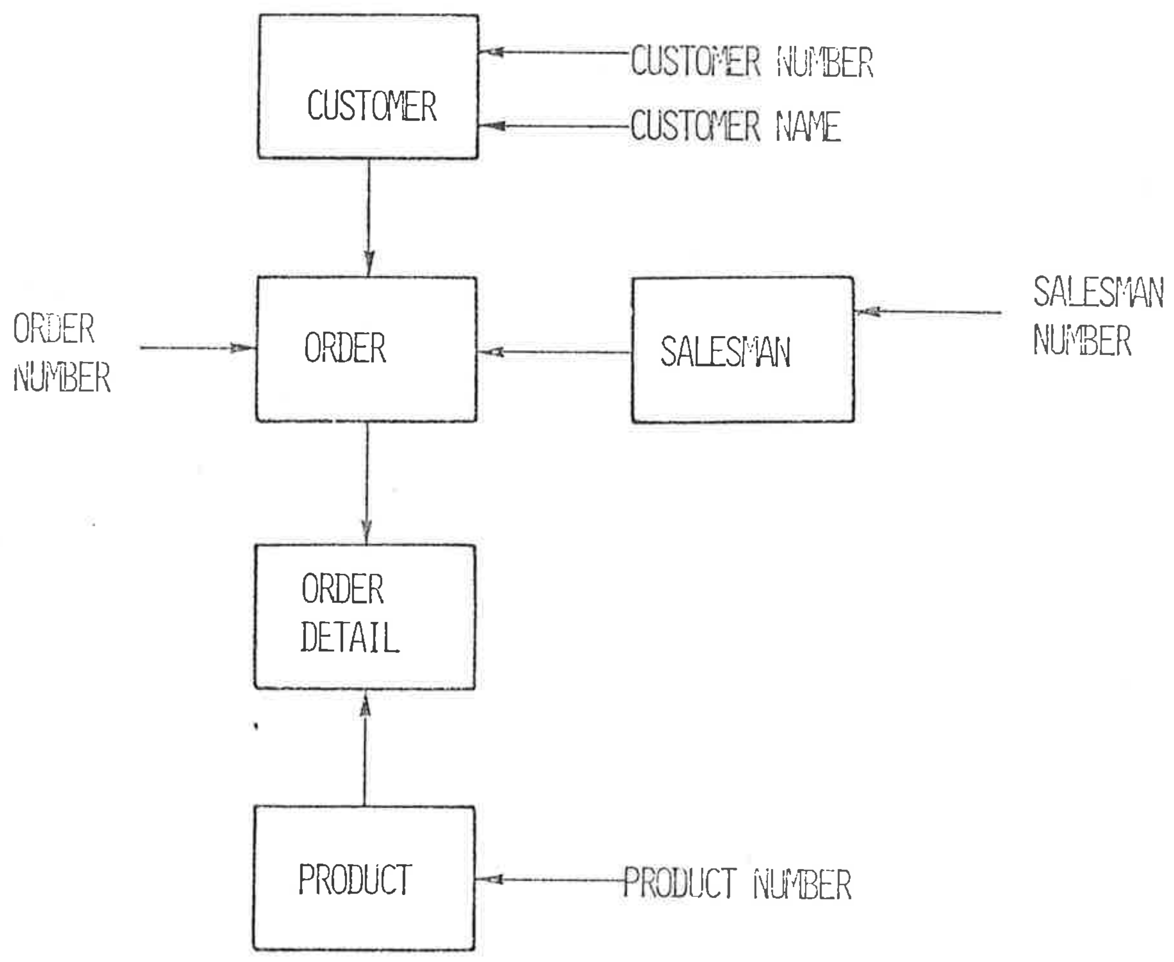
The conceptual model has been implemented using four record types: customer, order, order detail and product; three sets: customer-order, order-order detail, product-order detail; and three access keys: customer number, order number and product number.

The more comprehensive model where we have also included salesmen receiving orders, can be implemented in different ways depending upon how one is to use the system. We illustrate two possible ways below.

We first assume that we are only interested in finding the orders obtained by a given salesman, but have no need for any additional information concerning the salesman. We can achieve this by introducing the salesman number as an additional access key in the order record. We then obtain the following implementation structure.



If, however, we are interested in storing information about the salesmen, we can create a new record type , salesman, and relate this to the order records via a set salesman-order. We then obtain the following implementation structure:



3 CONSTRUCTION OF THE CONCEPTUAL MODEL

3.1 Identification

3.1.1 Identification of Objects

Objects can be defined as "units" which can be identified by their properties. Objects can be persons, things, events, etc. All objects of the same type (persons, for example) form an object class. The objects of a class have a number of properties, some of which are classified as "identifying properties". The latter are often aritificial properties given to the object to enable it to be identified.

Objects are identified by the name of the object class to which they belong (person, department) together with the names of the identifying properties.

The name of an object class and its identifying properties are put in a so-called object list. This could have the following appearance.

Name:	Identifying Properties:
Person	Person number, name
Department	Department number

3.1.2 Identification of Relations

A number of physical and/or logical connections will exist between the objects in the object list. These connections are represented as relations in the conceptual model.

One can distinguish between:

1. Relations between objects in the same object class. These will hereafter be referred to as "self relations". Here, when using the construction rules, we will confine ourselves to self relations between two objects of an object class.

2. Relations between objects from two different object classes. Where, unless otherwise stated, will be referred to as binary relations.

3. Relations between objects from three or more different object classes. These will be referred to as higher order relations.

The following approach can be used to decide what relations exist between objects in the conceptual model:

- the listed objects are compared two by two to find any binary relations. We choose to identify binary relations first because these are the most common and the easiest to identify (for example, "person" is employed in "department").

- for each object class one decides whether there exist any self relations between the objects in the class (for example, person "is subordinate to" person).

- where it is possible to find directly any higher order relations, these are noted (for example, an "order" is a relation between a customer, a product, and a salesman).

Although a relation between objects may have been identified, thi s does not preclude the objects from belonging to further relations. One object can belong to binary relations, self relations and higher order relations.

A salesman can have a relation "is employed in" to the department and a relation "order" to both customer and product.

Each relation is identified by the name of the relation class (for example, "is employed in") and by the identifying properties of the object classes to which it relates. The relation can, in addition, have its own identifying properties. One constructs a relations list which should contain the name of each relation class together with the names of its identifying properties and the names of the identifying properties of the object classes related.

An example of a relation list is:

Name:	Identifying Properties:	Objects Related:
Order	Order number	Salesman, customer, product
Employee	Employee number	Person, department

3.1.3 What are Objects and What are Relations?

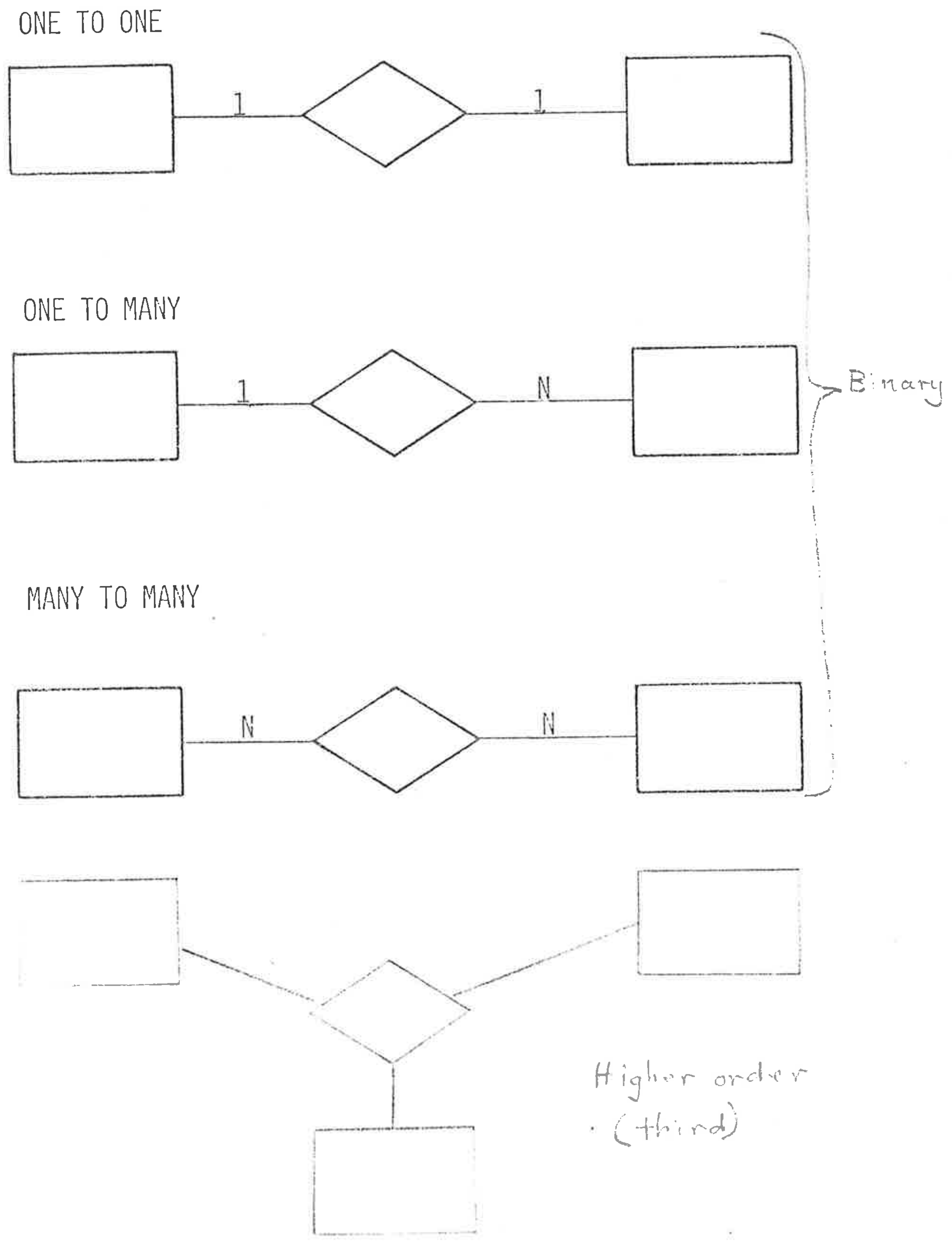
When analysing a given reality, it is often difficult to decide what are objects and what are relations. A number of relations are of such common usage that they are normally regarded as objects. An order, for example, can be a relation between a customer, a salesman, and a product.. An order will often be a relation which has its own identifying property (order number) and can occur in the model without one or more of the object classes it is usually a relation between. For example, order and product can appear in the model as two object classes with the relation "included in" between them.

In general, a relation which has its own identifying property and is of higher order can often be represented as an object in the model. This is particularly the case if the relation exists at the "edge" of the model, that is, one or more of the object classes related by the relation are not included in the model.

3.2 Construction of the Model

The notation introduced in Chapter 2 is used here to construct the conceptual model.

The various types of relation are represented as follows.



The model is constructed by drawing in each object class once and then filling in all the relations between these object classes.

As indicated in Figure 6a, the construction is an iterative process. In practice, one makes a list of some of the objects and relations, draws a picture of the model based on these, and then adds any additional objects or relations.

The name or description of all the object classes and relations in the model are written in. The construction of the conceptual model can be illustrated by an example.

Example:

The model is to contain all the employees of a firm, classified according to the department in which they work. It is also to contain information regarding who is subordinate to who in the firm. The object list will then be:

Name:	Identifying Properties:
Person	Person number, name
Department	Department number

Here we will have a relation between person and department, "is employed in", and the object class person will have a self relation "is subordinate to".

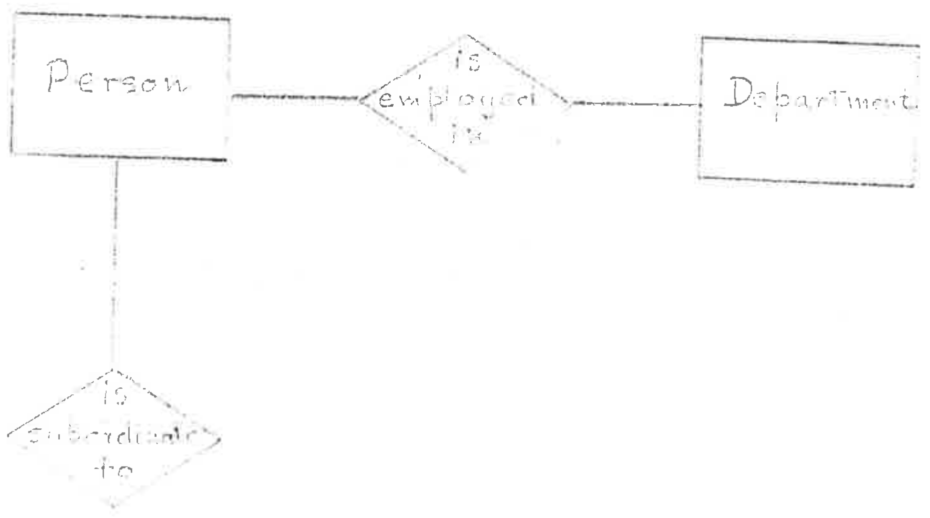
Relation list:

Name:	Identifying Property:	Objects involved in the relation.
is employed in	employment number	Person, department
is subordinate to		Person

When this part of the construction process is complete, we will have:

- a conceptual model based upon the given notation
- all objects and relations appearing in the model described by name and identifying property

the conceptual model is illustrated below



3.3 Types of Relation

The conceptual model, as far as it has been described, gives sufficient information to identify each object and relation class, and the order of the relations. In addition, it is advisable to know what type of relation we have in the model. Relations can be divided into three types:

1. One to one relations - 1:1.

   Each object in an object class is related to one other object in the same or a different object class. The converse is also true.

2. One to manyrelation or a 1:N relation.

   Each object in an object class is related to one or more objects in one or more other object classes, but the converse (i.e., regarding the first object class from an object in the other) is not true. If many objects in class B correspond to one object in class A, then only one object in class A will correspond to a given object in class B. Thus, the relation has an implicit direction, and one can also refer to it as an N:1 relation dependent upon the object class one chooses to view the relation from.

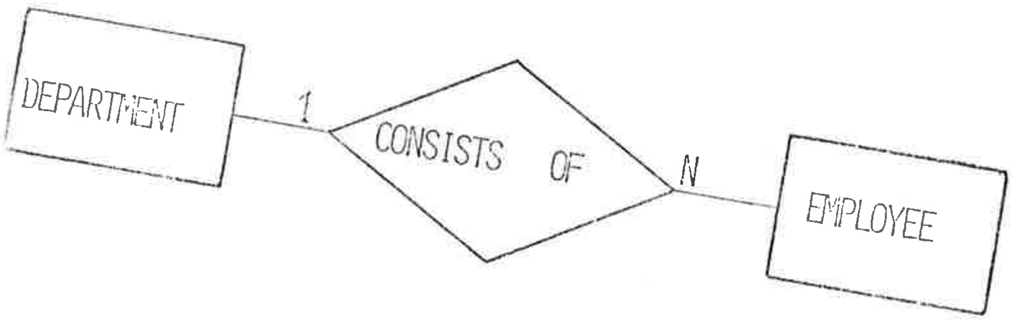
3. Many to many relations or N:N.

   To each object in an object class there corresponds one or more objects in one or more other object classes. The converse also holds. An object in class A will correspond to many objects in class B, and an object in B will correspond to many objects in A.

The relation type can be included in the model by writing it over the connecting lines between object and relation.

Example:

1:N



A department has many employees, but each employee belongs to only one department.

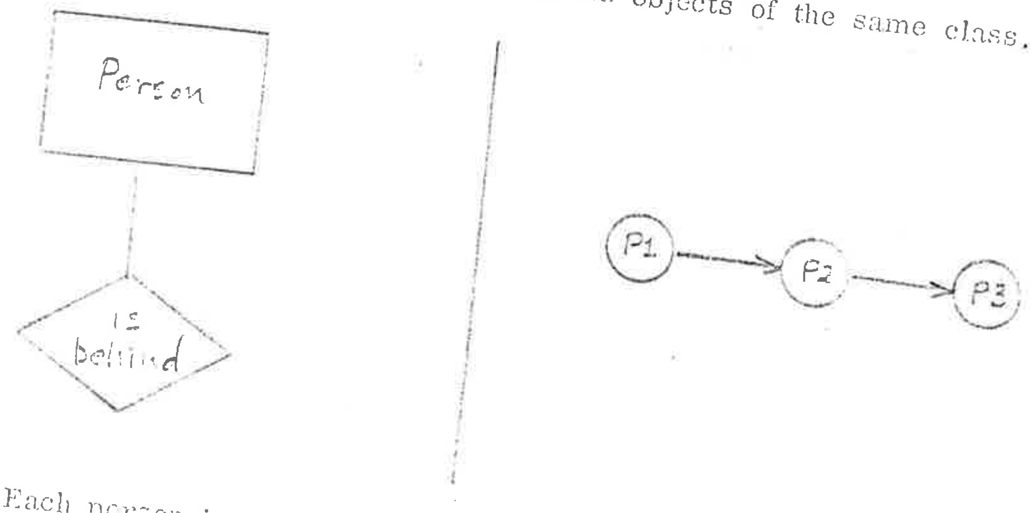
It is not always possible to represent the relation type in the manner shown above. For self relations and binary relations, no problems occur, however, for higher order relations the relation type is not so easily represented.

Here we will describe the three relation types for self relations and binary relations. An indication of how the type can be determined for certain higher order relations will also be given.

3.3.1 1:1 Relation

Self Relation:

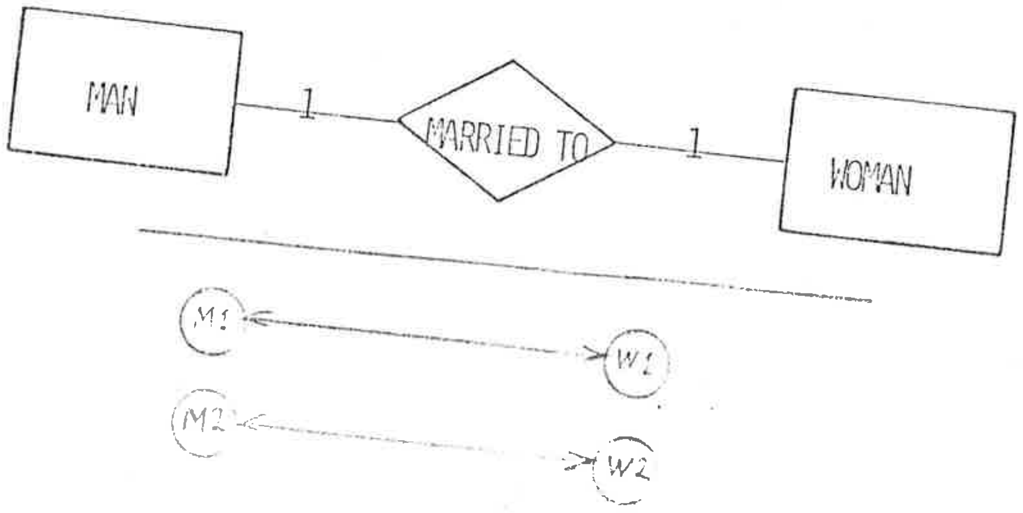
Gives a sequential structure between objects of the same class.



Each person in a queue stands "behind" only one other person in the queue.

Binary Relation:

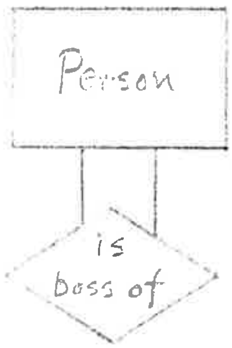
Represents pairs made up of one object in one class and another object in another class.

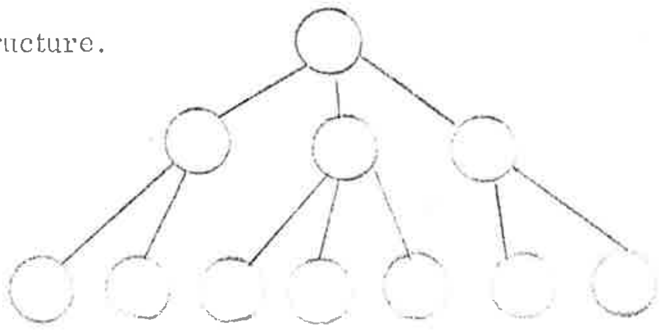


3.3.2 1:N Relation

Self Relation:

Represents a simple tree structure.

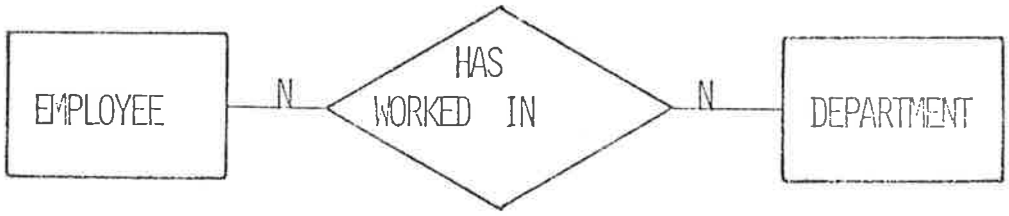




One person may be in charge of one or more persons, but each person is responsible to only one "boss".

Binary Relation:

Represents a two level tree structure.



Draw in the record structure yourself

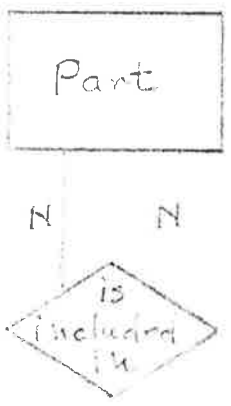
One department consists of a number of employees, but a person is only employed in one department.

3.3.3 N:N Relation

Self Relation:

Represents a simple net structure.

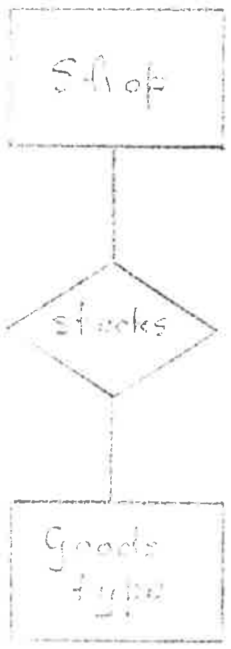
Draw in record structure



A product part can$\overset{\text{gets to}}{\wedge}$make up a number of other product parts while it itself consists of a number of product parts.

Binary Relation:

Represents a two level tree structure.



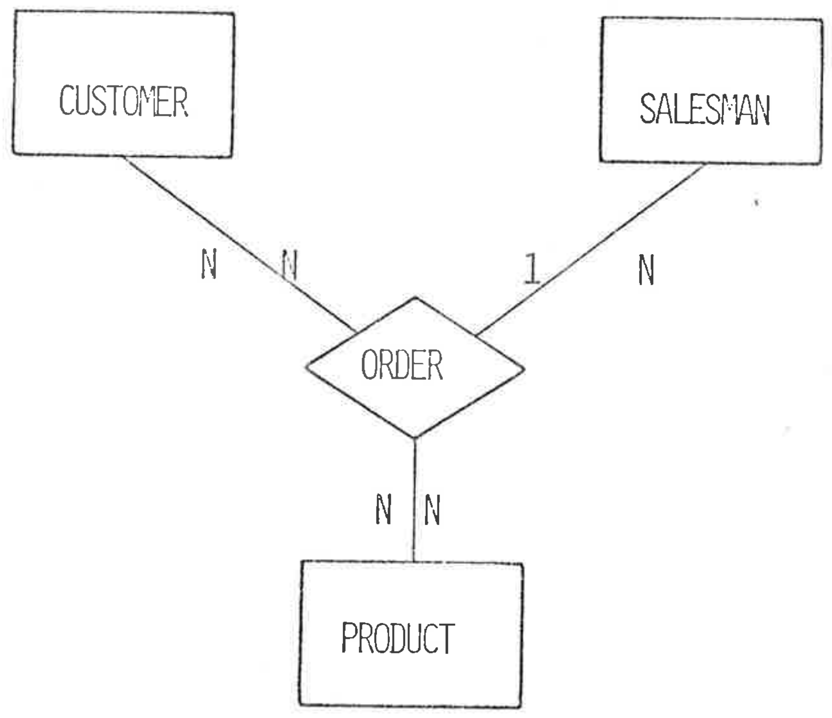
A shop stocks various articles or products, and each product can be stocked by a number of shops.

3.3.4 Higher Order Relations

For simple third order relations, it is possible to fix the relation type, or relation types (a higher order relation can in general have more than one type). This will be illustrated by two examples.

Example 1:

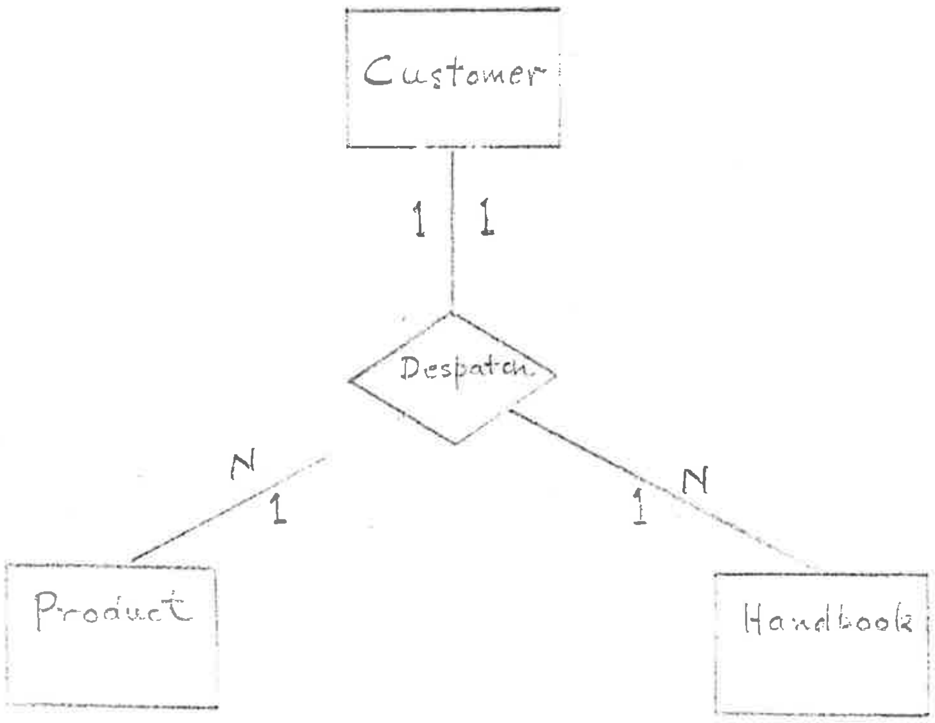
An order is a relation between a salesman, a customer, and a product.



A salesman has orders from many customers (1:N) whereas a customer deals with only one salesman. A customer has many products on order, and a given product is on order to many customers (N:N). A salesman accepts orders for many products, and each product can be ordered via a number of salesmen (N:N).

Example 2:

A model describes the despatching of products with their corresponding handbooks to customers. The object classes here are customer, product and handbook, and the relation between them is despatch.



To one customer, we can despatch many products (1:N) together with the corresponding number of handbooks (1:N), and each product will have associated with it one handbook (1:1).

When this part of the construction is completed the model will consist of:

- a drawing of the model using the given notation including (where possible) the relation types

- object lists and relation lists where the classes are named, and the identifying properties are given.

3.4 The Listing of Properties

For each object class and relation, the properties or attributes are collected. This will result in an expanded version of the object list. It is possible to distinguish between two types of properties:

- simple properties, that is, properties for which a given object or relation can have only one "value", e.g., name, age and height for the object "person".

- repeating properties, that is, properties for which the object or relation can have more than one value, e.g., examinations, previous employers, order details.

One can also indicate which simple properties are identifying properties. An identifying property need not necessarily have a unique value. For example, person number will normally be a uniquely identifying property, but a person's name will, or may, be ambiguous.

The object list can have the following appearance.

Name	Simple Properties:	Repeating Properties:
Person	Person number, name, address, pay	Schools, previous employers
Department	Department number, Department name	

The relation lists could be as follows:

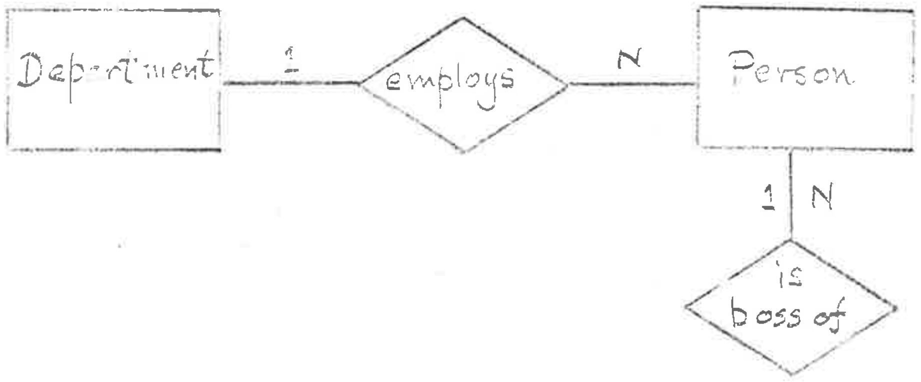
Description (Name):	Simple Properties:	Repeated Properties:	Related Object Classes:
Employed in	Employment number, work		Person/department

3.5 Testing the Information Content of the Model

The conceptual model (or diagram) plus the object and relation lists including the associated properties and attributes should now contain sufficient information to enable one to attempt to answer the questions which will be posed to the projected database. It is not easy to give systematic rules as to how such a control should be carried out. What one does is to set the questions to the model which will later become the application programs working against the database to see whether the model is satisfactory.

For each question, one must follow the structure of the conceptual model and determine whether each object or relation contains the required information. This procedure can be illustrated by an example:

The conceptual model:



The structure before testing the information content.

Object list:

Name:	Simple Properties:	Repeated Properties:
Department	Department number* Department name	
Person	Person number* Name, address, salary, work code	Schools Previous employers

Relation list:

Description:	Simple Properties:	Repeated Properties:	Objects Related:
Employed in	Employment number* Date employed		Person Department
Is in charge of			Person

The questions we wish to answer are:

1. Which people are employed in each department?

2. Find the pay, training and previous experience of each employee.

3. What kind of work and what responsibility each employee has?

4. Who is directly responsible to each employee?

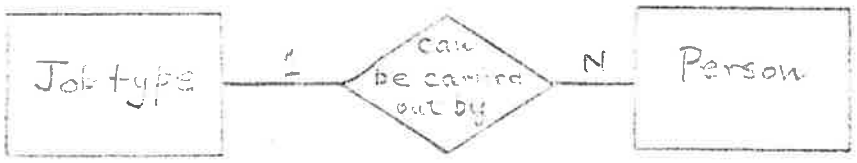
5. What is the functional connection between the departments ?

The answers to the questions using the conceptual model are given below:

1. For each department, the relation "employed in" is used to find all the employees working in the department.

2. For each employee, the object list contains information concerning his salary, education and previous experience.

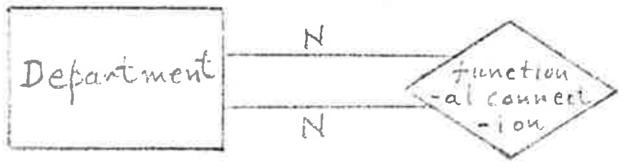
3. The object list for each employee contains a job code. To make available information as to what a given job involves, we must create a new object class called "job type", identified by the property job code, and a new relation "can be carried out by" between the new object class and the object class "person".



The new object class "job type" will contain information about the nature of the job and the responsibility it entails.

4. For each employee the relation "is in charge of" is used to find out which persons that employee has directly under him.

5. The organizational relation between the employees in each department is obtainable via the answer to question 4, and this is represented by a simple tree structure. The functional connections between the various departments will, in general, be more complicated, and must be represented using a net-work structure.



The relation list for "functional connection" must contain the properties necessary to describe the functional connection between the departments.

When the procedure of answering the given questions is completed, one must decide whether the conceptual model has the necessary information content. If some of the questions cannot be satisfactorily answered, then one must go back to the point in the construction rules where the necessary changes to the model can be made.

In the example, questions 1, 2 and 4 could be answered satisfactorily by the model. However, to answer the questions 3 and 5, the model had to be added to. This required a return to point 2.1 in the construction procedure.

The final appearance of the conceptual model for the example can be drawn in below.

The complete object list:

Name:	Simple Properties:	Repeated Properties:
Department	Department number* Department name	
Person	Person number*, name*, salary, work code	Schools, Previous employers
Work Type	Work code*, job description	Job responsibilities

The complete relation list:

Description:	Simple Properties:	Repeated Properties:	Objects Related:
Employed in	Employee number, D Date employed		Person, department
In charge of			Person
Organization	description		Department
Person responsible			Person, job type

When the above test is completed one will have arrived at the following:

- a conceptual model containing the names of all objects and relations together with the relation type for the majority of the relations

- a complete list of all objects and relations plus their attributes/properties.

3.6 Reduction of the Model

At this stage in the construction, the appearance of the conceptual model will, to a large extent, reflect the constructors intuitive understanding of the reality he wishes to represent. It is not possible to uniquely define the "best" conceptual model. A number of "rule of thumb" rules can, however, be given. The use of these will generally enable one to reduce the number of relations and/or objects in the model.

Such reductions will often be accompanied by the loss of some of the possibilities of the original model. Therefore, for each reduction rule, the way in which the information content may change will be given.

It is important to realize that the reduction rules do not automatically result in the best solution. Other factors which can weigh for or against reduction will often be involved. These may often be related to physical or practical considerations involving the actual database. This will be discussed in the description of the implementation structure. Factors involving the information content may also require consideration. It is not always sufficient to be able to answer a given question to the database and one should also consider how it is to be answered. This will have direct consequences for the application programmes to be run against the database.

The six reduction rules will be handled separately. Their order is random and the rules can be applied in an arbitrary order. The information content of that part of the model affected by a reduction must always be tested after the reduction. This is especially important for the possibilities which one loses. These are listed for each reduction rule.

3.6.1 Reduction of a Number of Binary Relations to One Third Order Relation

This reduction is possible if:

- binary relations between pairs of three object classes exist

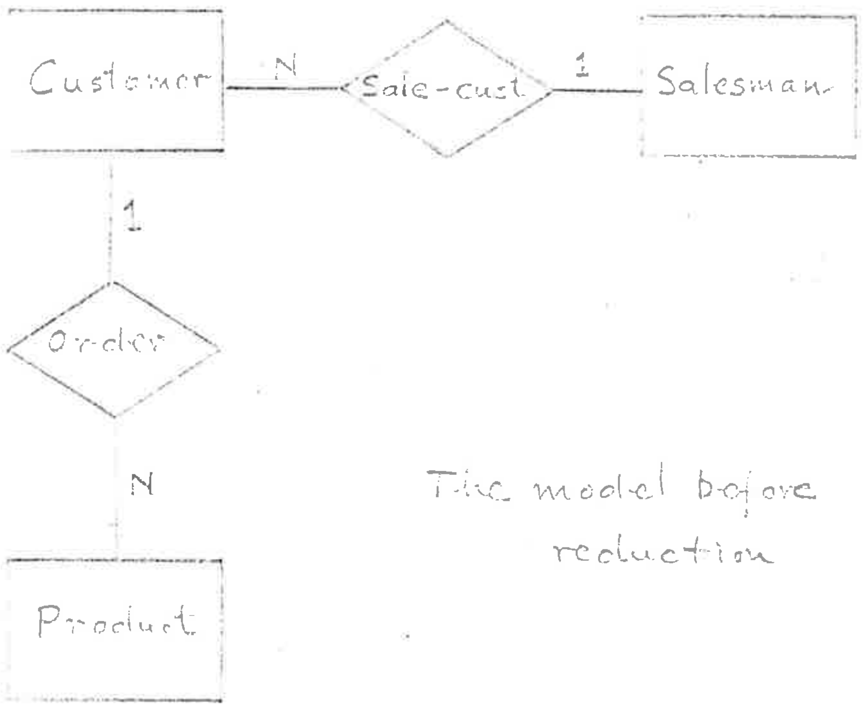
- the binary relations have a common property, that is, one property can be drawn out of each relation and put in the new relation.

The reduction involves integrating the binary relations in one third order relation. The type of the third order relation will not always be easy to determine and will depend upon from which object class the relation is viewed. The relation type for a third order relation can, for example, be 1:N between one pair of object classes, but N:N between another pair.

Two examples of this type of reduction are given below.

Example: 1

Part of a conceptual model consists of three object classes, customer, salesman, and product. The relation sale-cust exists between salesman and customer and indicates which customer a salesman has. Between customer and product, is the relation order which indicates which orders a customer has.

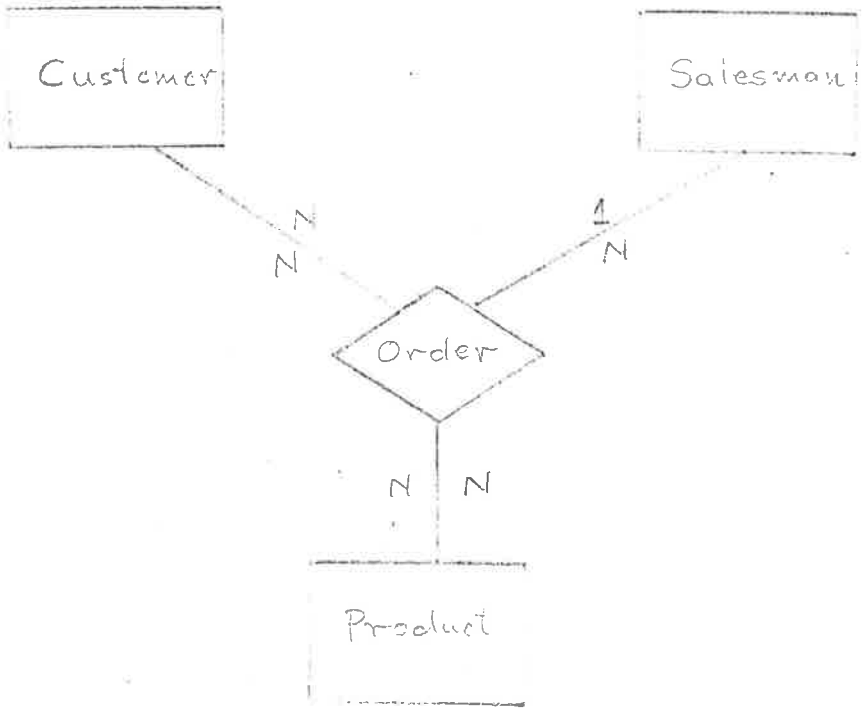


The model before reduction

The relations sale-cust and order are both 1:N binary relations and have in common that they both involve customer.

The two relations can be merged to a different third order relation called "order". This reduces the size of the model, and also improves its information content as it is now possible to find all the products a salesman has obtained orders for without going via the customers.

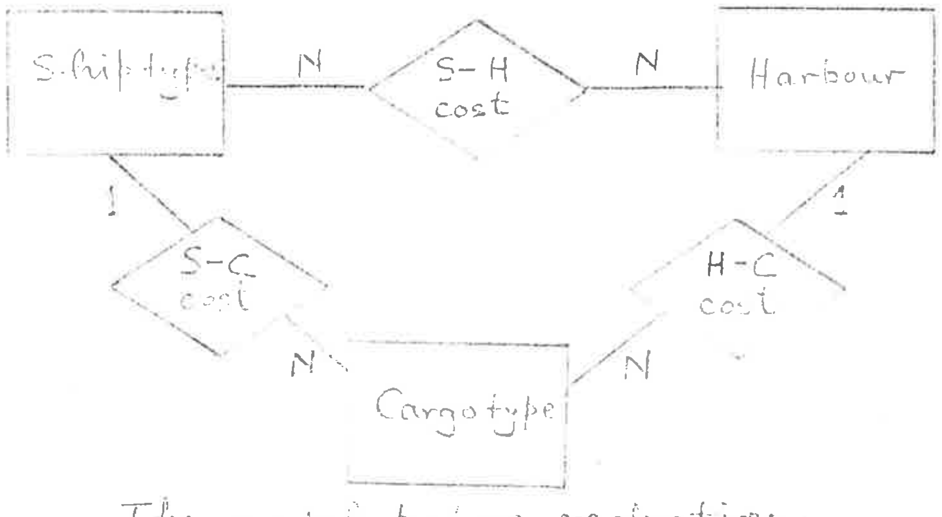
The model after reduction



Example 2

The previous example involved the reduction of two binary relations to one third order relation. The next example shows how three binary relations can be reduced to one third order relation. In contrast to the previous example, we have lost part of the information content as a result of the reduction.

The example describes the costs involved in loading and unloading cargo for ships. The object classes involved inthe determination of costs are: cargo type, ship type, and harbour.



The model before reduction

- The relation S-H-cost contains information about the cost of a given ship type to lie in a given harbour.
- The relation S-C-cost contains information about the cost of loading/unloading various cargo types for each ship type.
- The relation H-C-cost contains information about the cost of loading/unloading the various cargo types at each harbour.

S-H-cost is of the type N:N as each harbour can receive a number of ship types, and each ship type can lie in a number of harbours. S-C-cost and H-C-cost are both of type 1:N.

To calculate the cost of loading/unloading a given cargo type one must begin with the ship type and the harbour, and calculate the S-H-cost between these, the S-C-cost between ship and cargo type, and the H-C-cost between harbour and cargo type.

Had it been desirable to have a combined discount for certain combination of harbour, ship type and cargo type, then this cannot be directly represented by the model.

Should the model be reduced by combining the binary relations in one third order relation, then this relation must have properties which identify ship type, harbour and cargo type. Such a third order relation will also enable combination discounts involving two or all three of the object tasks to be expressed as properties or attributes of the third order relation "cost".

The reduced model can be drawn in below:

One should notice that the information content is somewhat altered in this reduced model. While the first model could give the cost for a given ship type to lie in a harbour directly, the S-H-cost will now be bound to the cargo type to be loaded or unloaded. Corresponding comments can be made for S-C-cost and H-C-cost. On the other hand, a combination discount can be included in "cost" directly, which was not possible for the original model.

3.6.2 The Reduction of a Number of Relations with an Object Class Inbetween, to One Relation

This reduction is possible if:

- a number of binary relations of type 1:N exist between one object class and surrounding object classes

- all 1:N relations have their "N" side towards the surrounded object class and their 1 side towards the surrounding object classes

In addition, the surrounded object must be an "abstract type" (refer to Section 2.1.1).

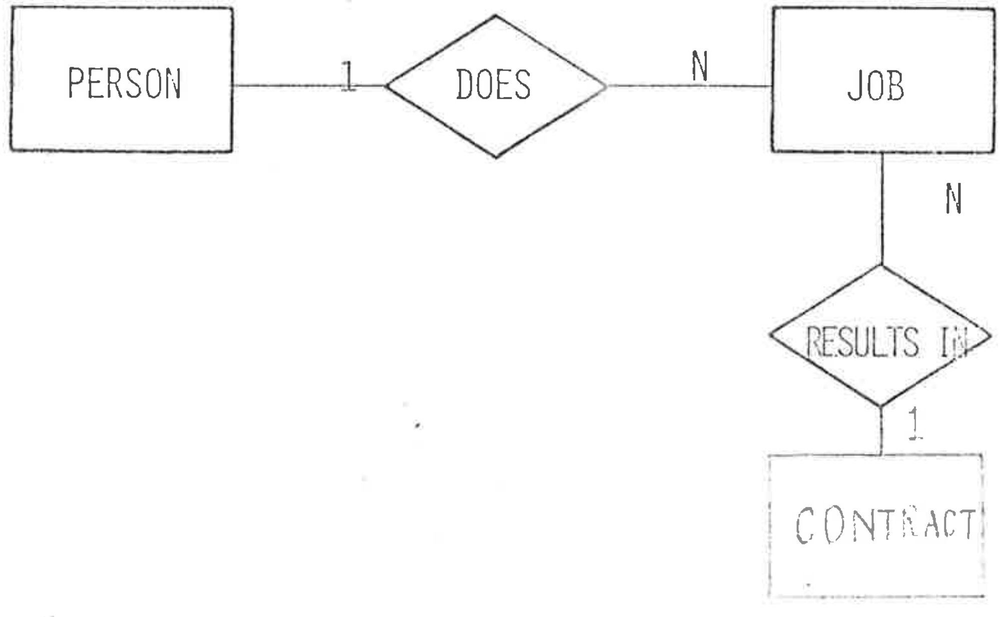
The reduction consists of combining the surrounded object class and all the relations joined to it by a single new relation. This reduction will usually result in changes in the information content of the model:

- some of the information previously contained in the surrounded object class may be moved to the surrounding object classes or to the new relation

- information paths which previously used the surrounded object class, will now go via the surrounding object classes.

An Example:

The model describes the connection between persons, jobs and contracts.

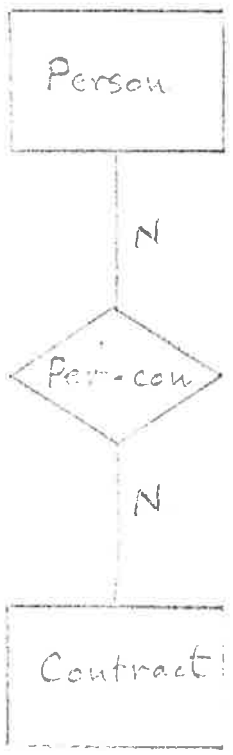


Here, job is the surrounded object class, with relations to person and contract. The relation "carries out" gives all the jobs a person is working on. The relation "involves" gives all the jobs a contract results in.

If the model is to be reduced, then some of the information in job will be moved to contract and person. In "contract", information about the types of job involved could be placed; which information about which types of job a person is qualified for could be moved to "person".

The new relation replacing job, "carries out", and "involves" will be of type N:N as one person can be involved in a number of contracts while one contract can involve a number of persons.

The reduced model.



The relation "per-con" can contain information about a person's position in relation to a contract (e.g. responsible for). Such information could not be represented in the original model since all communication between contract and person went via job.

3.6.3 The Merging of Two Identical or Nearly Identical Object Classes which are Joined by a Binary Relation

This is possible if:

- the two object classes are so similar that no great changes in the properties occur as a result of the combination

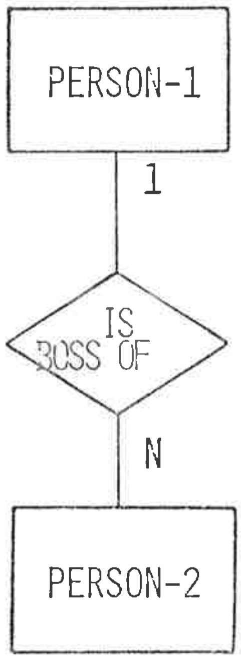
The reduction involves merging the two object classes to one, and replacing the binary relation by a "self relation". Information specific to one object class is put in the other object class.

The information which can be lost with this reduction is any information relating to the sequence of objects in the removed object class.

The reduction also removes a good deal of redundant information.

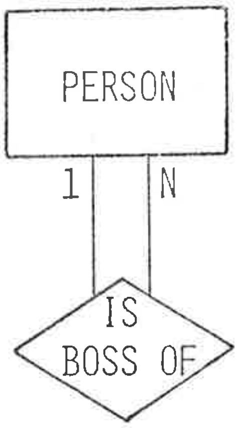
<u>Example 1:</u>

A model describes the hierachy between the employees in a concern.



"Person-1" contains information about each person. "Person-2" contains an identifying property for each person (redundant information). "Is in charge of" is the relation which joins a person to those persons under him.

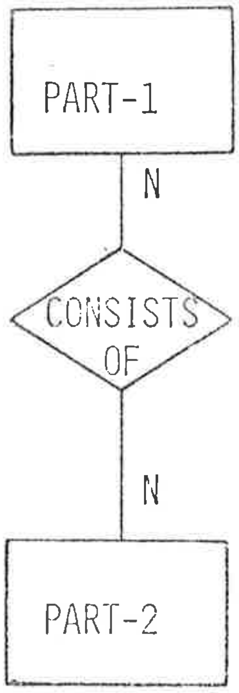
The model is reduced by replacing the two object classes by one, namely person. The relation "is in charge of" becomes a self relation, but will otherwise be unchanged. The information content will not change provided no special sequence existed for the objects in Person-2.



The reduced model

Example 2:

In this example, an N:N relation describing a product structure is taken.

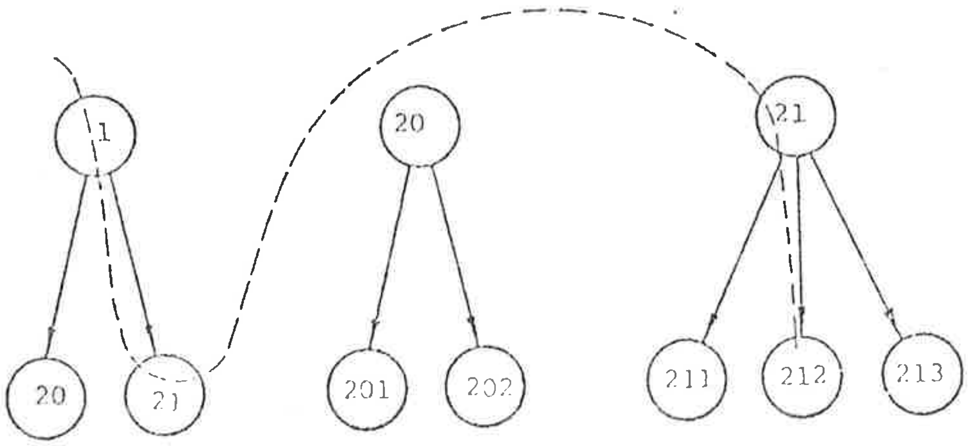


The relation "consists of" is N:N as one part can go towards the making of a number of other parts while it consists of a number of parts.

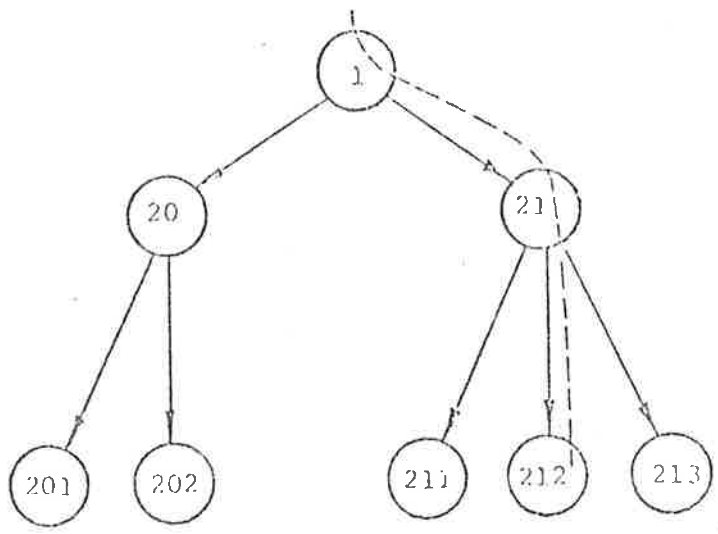
The reduction here consists of merging the two object classes as in the previous example, and changing "consists of" to a self relation.

The reduced model can be drawn as below.

For this type of reduction, (for both examples) the reduced model makes wandering about in the structure easier. If one, for example, wishes to go downwards in the hierachy in the first example, one would have to return to person-1 for each downward step in the structure. That is, after finding person-2 one would then find the corresponding object in person-2 to continue the searching. The dotted line shows how one can arrive at person 212.



After reducing the model, the searching can go directly through the tree structure. The dotted line again shows how we arrive at person 212.



One will also remove a good deal of redundant information as all objects not at the top or bottom of the hierachy will be represented twice in
the unreduced model as opposed to once in the reduced model.

3.6.4 Two Symetrical 1:N Relations are Reduced to one N:N Relations

This reduction is possible if:

- two 1:N relations go in opposite directions between two object classes

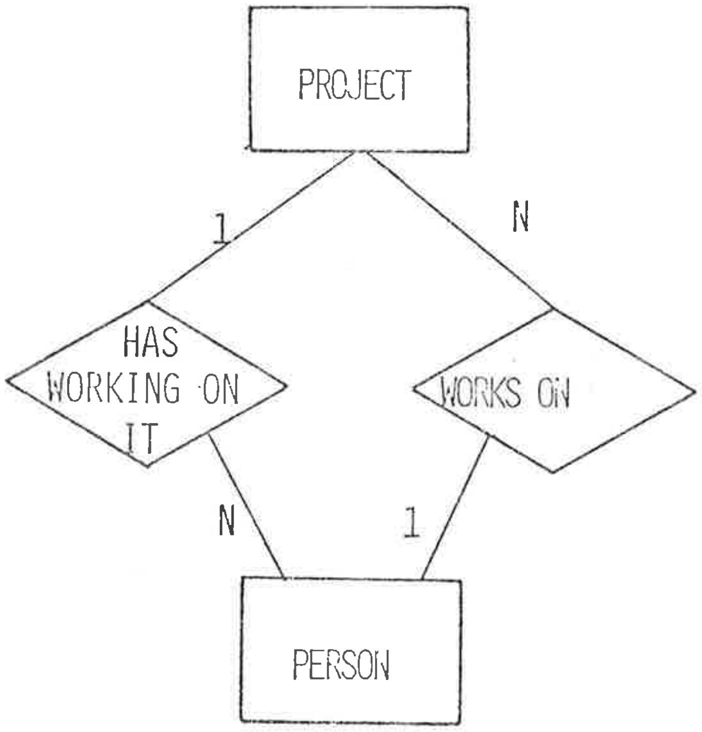
- the relations have an information content which can be combined in one relation

The reduction involves merging the two relations into one.

As long as the new relation contains all the properties found in the two 1:N relations, no loss of information will occur as a result of the reduction.

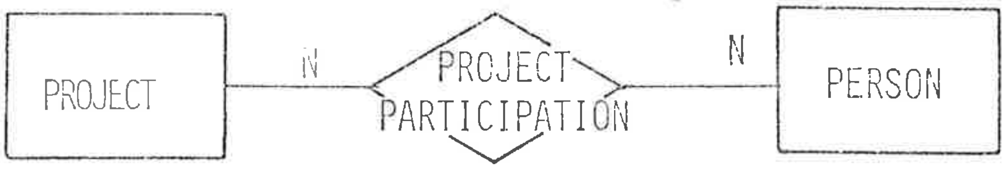
Example:

A model describes projects and people working on projects. Two 1:N relations, works hours on, and has workers exists between the object classes person and project.



The model indicates that one should be able to find all the projects a person is working on and all the persons working on a given project.

The model is reduced by merging the two relations to one single relation of type N:N.



"Project participation" satisfies all the criteria covered by the original two relations.

The properties belonging to the two previous relations can be given to the new relation "project participation". One could also here add information about the work load upon each person for each project he participates in.

3.6.5 A 1:N Relation Between Two Object Classes is Reduced to a Repeating Group for the Owner Object Class

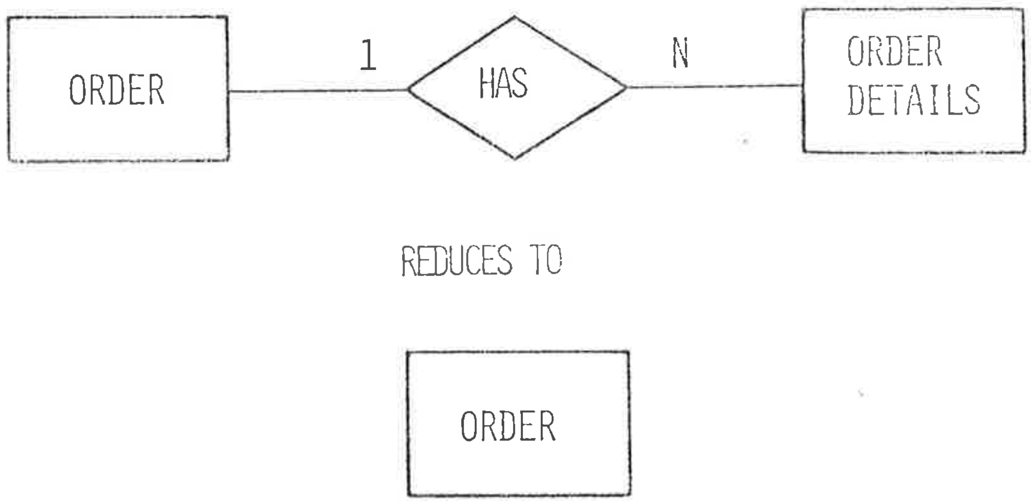
This reduction is possible if:

- a 1:N relation exists between two object classes
- the object class on the N side is not related to any other object classes
- the properties belonging to the relation and the object class on the N side can be moved to the object class on the 1 side of the relation.

The reduction is achieved by removing the relation and the N side object class and putting all the properties of the relation and object class to be removed in as repeating groups of properties in the 1 side object class.

Example:

A model describes orders and all the order details making up an order.



The object list for order will be extended by all the properties of the original order detail object class plus any properties of the relation "has".

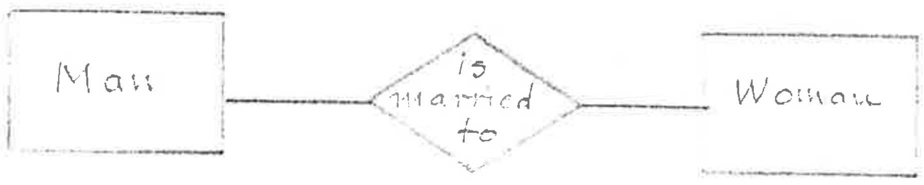
No change in the information content need occur here.

3.6.6 A 1:1 Relation Between Two Object Classes is Removed and the Two Object Classes are Combined into One

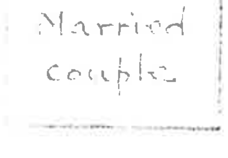
This reduction can always be carried out when a 1:1 relation exists between two object classes. It will not result in any loss of information. The reduction is achieved by removing the one object class and moving all properties of the removed class to the remaining object class. In addition, any properties belonging to the relation will also be moved. If the removed object class had other relations to yet further object classes, these relations will be transferred to the new object class.

Example:

A model describes men, women, and who is "married to" who.



The reduction results in a new object class called "married couple", which contains all the properties before found in the men and women object classes together with any from the relation "married to".



When all possible or relevant reductions have been made to the conceptual model one will have:

- a sketch of the model containing all the object classes and relations together with the relation type for the majority of relations. This model should satisfy the revelent information needs as best possible

- a complete list of the object classes and relations together with all their properties

- a description of the information needs one expects the model to satisfy

At this stage of the construction of the database, one has a documentation of the logical content of the database together with a colloquial (high level) description of the application programs which are to run against the database.

4 CHOICE OF IMPLEMENTATION STRUCTURE

The previous chapter describes how one with the use of the concepts of object and relation can describe reality by a conceptual model. This model is constructed mainly from a knowledge of that part of reality one wishes to represent and of its required information contant.

To implement the model further information concerning the way in which and how often the information in the model is to be used is required. In addition one must be aware of the characteristics of the mechanisms used in practice.

In the introduction the phases one should go through when planning the implementation structure were described as a sequence. This sequence need not be followed rigourously, and should be regarded as a guide to the main activities leading to an implementation.

Before describing the individual phases some general characteristics of practial usage and mechanisms will be described.

4.1 Storage Mechanisms

We distinguish between the following storage mechanisms:

- Random (hashing)
- Sequential
- Serial
- Proximity

The relevant criteria upon which a choice should be based are:

- Initial storage
- Storing of new records
- Removal of records
- Space usage, use of freed space
- Sensitivity to changes in assumptions made.
- Information retrieval
- Storage order

The storage mechanisms can be found implemented in various ways. We give the general characteristics of each.

4.1.1 Random

In a random file the records are "arbitrarily" stored depending upon the value of some data element which is used as a key such that the initial storage of records (loading the data base) can result in many accesses to the same part of a file. (this can be avoided if one knows the randomising algorithm and sorts the records to be stored such that their order corresponds to the physical order they will be stored in on the file).

The storage of new records on a hashing file is normally a fast operation which does not affect the other records on the file. The same holds for the deletion of records.

The use of the space on the file depends upon how the values of key are distributed by the randomising algorithm. Usually it is not possible to attain a usage of space better than 70-90% before the file becomes less effective in use. A randomised file normally requires the allocation of the same amount of space independently of whether it holds few or many records.

A hashing file is sensitive to changes in the assumptions about the values of the key and the maximum number of records which can be stored.

A hashing file gives fast access to the individual records, and the access time is alomost independant ot the total number of records which can be stored on the file.

The table on the next page can be used in estimating the size requirements for randomised and assumes that the values of the key have a poisson distribution. The file is divided into a main area and an overflow area. Each address (bucket) in the main area has room for a number of records. The load factor of a hashing file is the ratio of the total number of records stored in the file to the number of records stored in the main area.

The table gives the percentage of records which go into the overflow area as a function of the load factor and the number of records per bucket.

For example if a bucket holds 10 records and a load factor of 0.8 taken, 5,32% of the stored records will lie in the over- Thus one might that it is best to choose a large bucket size, but this will increase the search time in each bucket.

Records Per Bucket in Main Area:	Load Factor: 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	4.84	9.37	13.61	17.59	21.31	24.80	28.03	31.17	34.06
2	0.60	2.19	4.49	7.27	10.36	13.65	17.03	20.43	23.79
3	0.09	0.63	1.80	3.61	5.99	8.82	11.93	15.37	18.87
4	0.02	0.20	0.79	1.96	3.76	6.15	9.05	12.32	15.86
5	0.00	0.07	0.37	1.12	2.48	4.49	7.11	10.26	13.78
6	0.00	0.02	0.18	0.67	1.69	3.38	5.75	8.75	12.24
7	0.00	0.01	0.09	0.41	1.18	2.60	4.74	7.60	11.04
8	0.00	0.00	0.05	0.25	0.84	2.03	3.97	0.08	10.07
9	0.00	0.00	0.02	0.16	0.61	1.61	3.30	5.94	9.27
10	0.00	0.00	0.01	0.10	0.44	1.29	2.88	5.22	2.53
11	0.00	0.00	0.01	0.07	0.33	1.04	2.48	4.80	2.01
12	0.00	0.00	0.00	0.04	0.24	0.85	2.15	4.36	7.51
14	0.00	0.00	0.00	0.02	0.14	0.57	1.65	3.84	6.67
16	0.00	0.00	0.00	0.01	0.03	0.39	1.23	3.08	6.00
18	0.00	0.00	0.00	0.00	0.05	0.28	1.01	2.65	5.45
20	0.00	0.00	0.00	0.00	0.03	0.20	0.81	2.30	4.99
25	0.00	0.00	0.00	0.00	0.01	0.09	0.43	1.65	4.10
30	0.00	0.00	0.00	0.00	0.00	0.04	0.20	1.28	3.47
35	0.00	0.00	0.00	0.00	0.00	0.02	0.18	0.94	2.98
40	0.00	0.00	0.00	0.00	0.00	0.01	0.12	0.73	2.60
50	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.45	2.04
60	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.30	1.65
70	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.20	1.37
80	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.13	1.14
90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.97
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.83

The % age of records in the overflow area as a function of the load factor and the bucket size (in records).

The physical sizes such as the block size of a storage medium and buffer sizes in core will also set practical limits upon bucket sizes. Some of the advantages are lost if buckets cannot be read into core in one physical access.

4.1.2 Sequential

In a sequential file the records are stored in order according to the value of some key. For initial storage the records can be correspondingly sorted prior to loading, and this will then be a quick operation.

The storage of new records requires either the moving of all records "behind" the new record, or the use of an overflow area when the physical ordering will be partly damaged until the file can be reorganised.

The deletion of records will give "holes" in the file which can be used if a new record belonging to the hole arrives, or if one moves all the records behind the hole.

Space exploitation in a sequential file will be close to 100% especially if freed space can be used.

A sequential file will be sensitive to changes in the assumptions made with regard to the storage and deletion of records. If such "traffic" becomes considerably greater than expected the accessing of records will be degraded, or one will need to reorganize the file more frequently than originally planned.

A sequential file will be expensive to use if one is mainly interested in only a small number of the records in the file.

4.1.3 Serial

In a serial file the records are stored in the order in which they arrive at the file. Both the initial storage and the storage of new records will be quick operations. The removal of records will also go quickly provided one does not "compress" the file.

Space usage will be nearly 100% as long as freed space can be reused.

A serial file will normally require at least one access mechanism.

4.1.4 Proximity

For this storage mechanism records are stored near one another an the physical storage medium depending upon some criterian which could for example be the value of a data element. A new record will be stored as closely as is possible to the records to which it "belongs" without moving any other record.

This storage mechanism will have much in common with the serial, but will in addition use the possibility of providing an access path via the physical ordering of the records.

4.2 Access Mechanisms

We distinguish between the following access mechanisms

- Random
- Index
- Chaining
- Searching

The criteria which should be used when choosing access mechanisms are as follows:

- Space requirements
- Fetching records
  Large/small fraction of the total number sorting
- Storing records
  Initial storage
  Storing of new records
- The removal of records
- Sensitivity of changes in assumptions

As with storage mechanisms, access mechanisms of various kinds exist. The general characteristics of each access mechanism will be given.

The various mechanisms do not exclude one another. It is fully possible, in principle, to access the same record via hashing, via one or more indexes, via one or more chains or via searching.

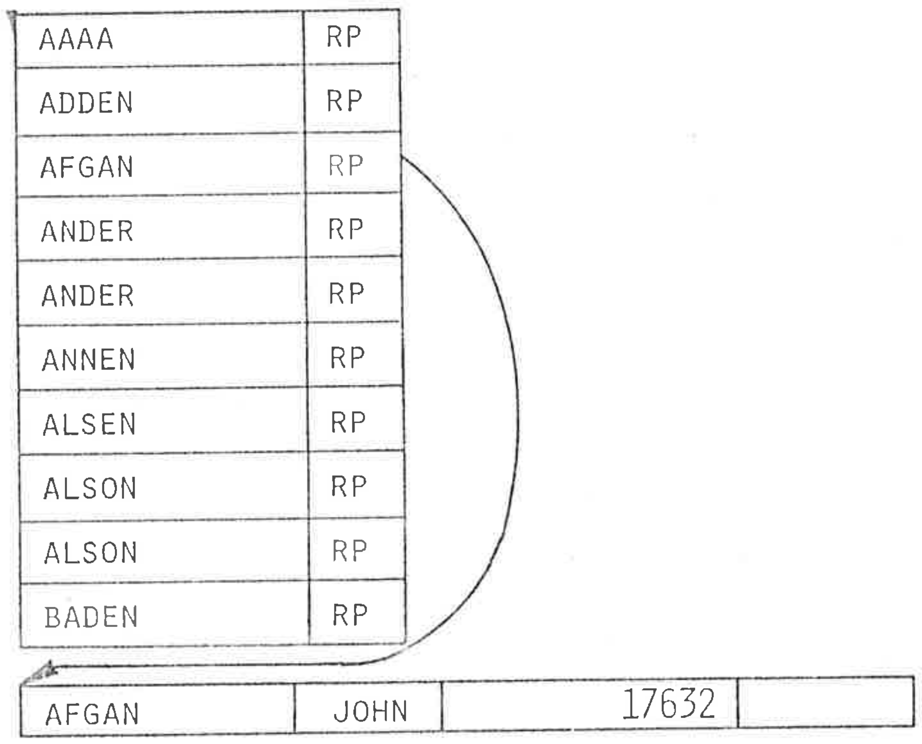
4.2.1 Random

Most of what applies to hashing as a storage mechanism applies to it also as an access mechanism.

The advantages of using hashing as an access mechanism are that it requires no extra space, and no table or pointers need to be updated when a record is stored or deleted.

4.2.2 Index

An index table is a table containing the value of the access key, and the address of the record, for each record indexed



The address of a record is found by looking up the value of the access key in the index table. The index table can itself be regarded as a file and can be organised in various ways just as can other files. It can be randomised (often referred to as a hash table), serial or sequential, (one may come across the term "index sequential file". As we have used the terms, and distinguished between a files organisation and its access mechanisms. an index sequential file will be a file which is organised sequentially and which has an index table as access mechanism).

Sometimes an index table will be organised as a combination of a number of physical organisations and access mechanisms. For example it can be divided up into a number of small tables where the entries in each table are stored sequentially, while the tables are stored serially and chained together in a logical tree structure. Searching in the table starts at the top of the tree.

An index table normally holds the value of the access key and the address of each record stored in the file. The space requirements for an index table are dependent upon the length of the access key, the length of an address, and the way in which the table is organised.

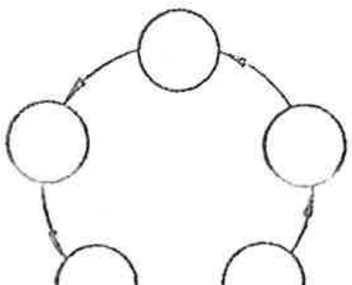
The time required to access a record via an index table is dependent upon how the table is organised and how the record is addressed from the index table entry. This can be either a direct physical address to the record, or the address of an area containing a number of records which must then be searched for the relevant record.

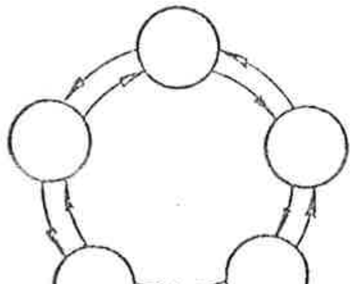
If the index is organised sequentially it can be used to fetch the records sequentially. This also gives one the possibility of fetching a selection of records for which the values of the access key lie between two given limits.

When storing or removing records from the file, the corresponding entries in the index table must be added or removed.

4.2.3 Chaining

As the third access mechanism we have the chaining together of records. This is acheived by storeing the address of the next record in the chain, and sometimes the address of the previous record, in each record. For single chains only the address of the next record is stored, while for double chains the addresses of both the next and the previous record are stored in each record.


Single chain


Double chain

Chaining requires space for the storing of addresses in each record. The space requirements depend upon the length of an address.

When accessing the records in a chain we fetch them one at a time which can require one physical access per record if the chained records are not stored close to one another.

When records are stored in a chain, pointers in one or more records must be updated. For single chains, one record, and for double chains two other records, must be updated. The same applies for the deletion of a record. (In some systems which use chaining the records are not physically removed from the chains under normal operation but marked as "dead", and removed physically in a reorganisation process). The records containing the address of the record to be removed are the ones to be updated. This is an easy matter for double chains where the record to be deleted holds the addresses of the records to be updated. For single chains however, one may have to go round the whole chain to find the previous record, and this will be costly if the chains are long.

4.2.4 Searching

As the last access mechanism to be considered we have searching. In principle one can find any record in a file by searching through it.

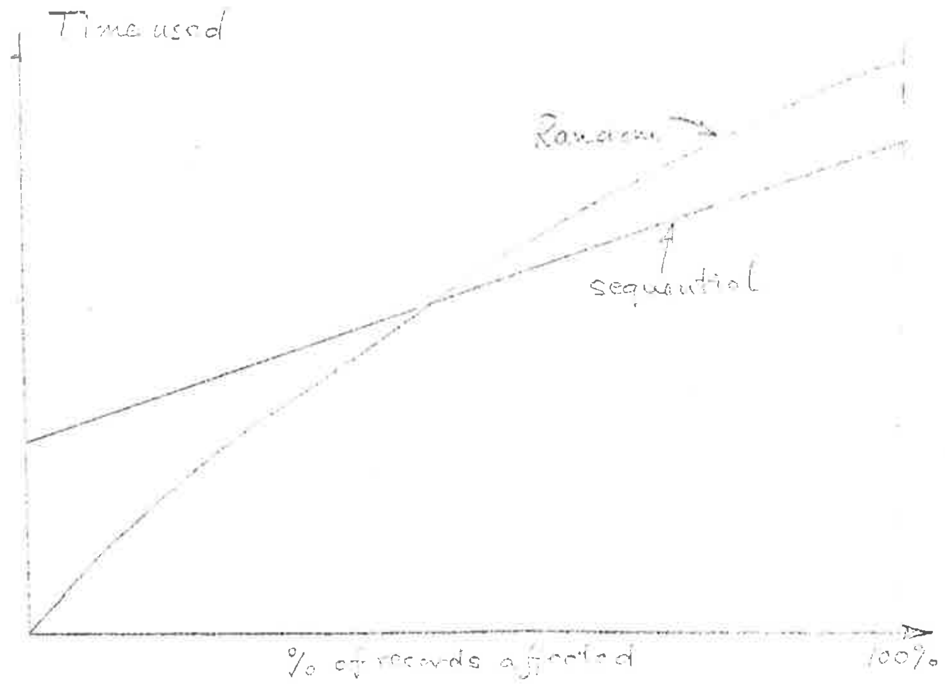
In an unsorted file one will on average have to search through half of the file to get a record which is in the file, but one must search the whole file to determine that a record does not exist.

If a number of records are to be fetched at the same time, these can be obtained by one search through the file thus reducing the access time per record.

A sorted file has the same characteristics as an unsorted file when searching for a single record, provided one starts at the beginning of the file.. However it will be faster in determining whether a record exists or not.

If a sorted file is stored on a directly addressable storage medium, then binary and other search methods can be used to access a record more quickly.

Searching is most advantageous when updating or accessing a large part of the records in a file. The relation between the access time per record for sequential and hashing files as a function of what fraction of the records are to be accessed is shown below.



4.3 The Grouping of Information

In the construction of the conceptual model information has been sorted into groups according to where it logically belongs and how it is to be identified upon retrieval.

In constructing the implementation structure one uses this as a starting point and may merge or divide the information groups to form records depending upon how the needs for exploiting resources, flexibility and protection against loss or misuse are to be satisfied.

When choosing the content of the various record types the following points should be considered.

- Data volume (number of machine words, bytes, characters) per object or relation
- Variations in this volume for the objects or relations of the same class
- The frequency of retrieval and updating of the various data elements
- Machine dependent sizes (block sizes on disk, buffer sizes in core etc.)
- Demands for security of the various data elements with respect to loss or misuse.
- The way in which the data is to be retrieved.

Normally the record sizes which it itpossible or sensible to have will be limited by the available hardware or software. There may often be a higher limit and perhaps a lower limit for the possible record sizes, and demands for space utilization, buffer sizes, processing effectivity and physical block sizes mean that information must be grouped in such a way as to give suitable record sizes.

The data volume (measured in machine words), bytes or characters) may well vary for objects or relations belonging to the same class. If this variation is large it may, for the sake of space utilization be advisable to use variable record lengths, or to spread the data over more than one record type. Thus information belonging to an object or relation may be stored in one or more record types depending upon how much there is.

The various data elements belonging to an object or relation may often be used with rather different frequencies. If one takes account of this and sorts the data elements into a number of record types according to their usage, one can obtain a faster access to the most frequently used and thus improve the total effectiveness of the system.

Both space usage and access time will as a rule be improved if account is taken of the characteristics of the storage medium such that records for example do not run over block boundaries.

In addition to the points mentioned above, the division into record types and files can be affected by demands for security as mentioned in 3.1.3.

4.4 Usage Patterns

By usage patterns we mean the way in which the database is to be used. The various ways in which a database can be used can be represented under three main headings:

- the demand for information retrieval
- the demand for updating
- the demand for security, integrity

4.4.1 Information Retrieval

Information retrieval together with updating will define the work load to which the database will be subjected. The most important parameters in connection with information retrieval are:

- transaction frequency
- required response times
- percentage of records affected/used per transation
- required sorting or ordering

Transaction frequency is the number of requests coming to the database per unit time. For batch use we are interested in how often a given program is run against the database, for example a report may be required once a week. For on line we need to know the number of request transations per unit time.

The response time is the time from when a program/transaction starts to the time when the output is available on the line printer/terminal.

The response time will not only depend upon the database structure, but also upon the actual work load on the machine.

The number of records accessed or used as a percentage of the total number of records in a file (or the whole database if this is not subdivided into files) will affect the choice of file organisation and access mechanisms. Thus a large percentage indicates that searching the whole data volume is best, while a small percentage is better served by some direct access mechanism.

If the records are to be retrieved in a definate order it will be most advantageous to have a sequential organisation.

4.4.2 Updating and Storing Records

Updating involves the retrieval of records, but in addition to the considerations then applying, one must also take account of:

- the frequency with which records are added or removed in number of records per unit time.
- which data elements are changed in updating

To estimate how much load one will get as a result of updating one must distinguish between the updating of "ordinary" data elements and those data elements which are used when storing or accessing a record. For the latter more time is required as the index table or pointers need to be changed. or the records physical position on the file may be altered in addition to the updating of the data element.

4.4.3 Security, Misuse

In addition to the demands for the minimisation of the resources required, the normal day to day usage of a database will have to be protected against loss of all or part of the database and against misuse of the data. These are often regarded as additional demands not affecting the choice of database structure in any way. However, a good deal can be done, while the structure of the database is being determined, to simplify the steps necessary to obtain the required protection against loss or misuse of data.

Such demands should therefore be decided upon and described early enough to be considered under the construction of the database.

If certain personal data are regarded as unsuitable for disclosure to unauthorized persons, these could be collected in one record type thus isolating the problem. Data redundance can be used as an insurance against the loss of data.

4.5 The Choice of Structure and Mechanisms

In the previous section the general characteristics of the various access and storage mechanisms have been discussed. We will below show how one can use the various mechanisms to implement a conceptual model.

4.5.1 Concepts and Symbols

To describe an implementation structure we use the concepts of record type, access key and set with their corresponding symbols as described in chapter 1.

4.5.2 The Representation of Objects and Symbols

Chapter 2 described how one, with the help of the concepts object, relation and identificaton, can make a model of reality. This model is based upon a knowledge of the data it must contain in order to represent reality well enough to satisfy the information needs.

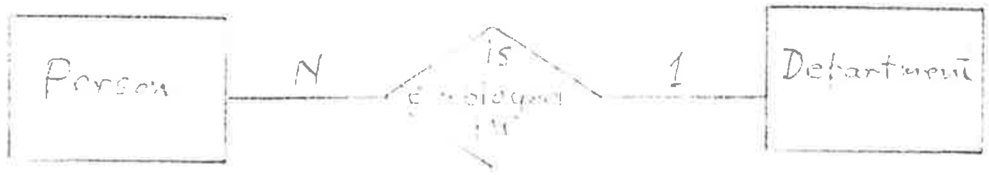
The conceptual model can be implemented in various ways depending upon how it is to be used and what mechanisms are available.

As a start we can say that we will describe each objecttype and relation type in the conceptual model by a record type in the implementation structure. Such an implementation will give a complete coverage of the information represented in the conceptual model, but will often be somewhat overdimentioned and unneccessarily costly in practice.

We shall indicate various possibilities in the following example.

Example

In chapter 2 a conceptual model of an employees departmental status was discussed. This conceptual model is shown below.



This model can be implemented using three record types.

- Employee record type. Each record describes an employee. It contains a data element which identifies the employee and various data describing him.

- Department record type. Each record describes a department and contains a data element identifying the department and various data describing the department.

- "Is employed in" record type. Each record describes an employment condition and contains the identification of the employee, the identification of the department, and various other data such as for example the date the person was employed.

We will then have implemented the conceptual model by taking account only of the information content. We could decide to organise the three record types as three sequential files:

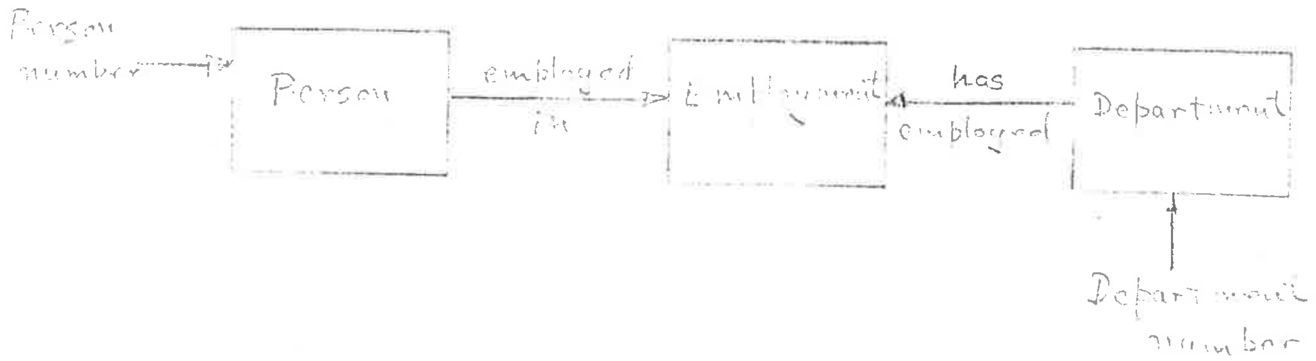
- An employee file sorted an employee identification

- A department file sorted an department number.

- An employment file sorted an department number as a primary key and employee identification as secondary key. (Draw the implementation structure)

This implementation will be able to "answer" all the questions set to the conceptual model and will be quite effective in producing a list of all departments where one gets the employee number and date of employment written out for each department. It will be less effective in producing a list over all employees and which departments they work in.

We can instead of organising the files sequentially establish access direct mechanisms as for example shown below.

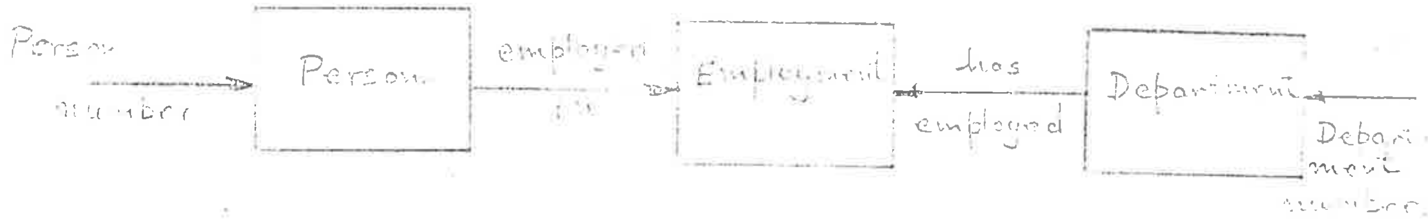


We can now go directly in to the employee file using employee number, the department file using department number, and the employment file using department number or employee number.

This implementation will probably be faster than the previous one in answering questions on who is employed in a department, or where an employee works but it will not be quicker in preparing a list of all departments with employee numbers and dates of employment.

In addition this implementation will probably require more space than the previous because of the access mechanisms.

We can instead of using two access mechanisms on the employment file join it to the two other files by using two sets. We than get the implementation structure shown below.



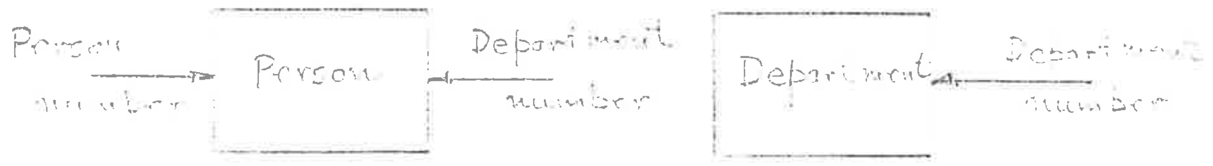
This implementation will have much the same characteristics as that shown on the previous page.

In the above implementation each object and relation has been implemented as one record type. It may often be advantageous to implement an object or relation as a number of recordtypes. Reasons for this can be that some information is required more often than other, or that the amount of information varies from record occurence to record occurence, or that some information is confidential and therefore best put in its own record type where it can be more easily protected against misuse.

The implementations shown above do not take account of the fact that the relation "is employed in" is a 1:N relation, i.e. an employee can only work in one department. The implementation is "over dimensioned" in that the relation record type can represent an N:N relation.

If we take account of this we can rationalise the implementation without either losing information or affecting the way in which questions can be answered.

Since an employee can only be in one department we can move information related to his employment into the employee record from the "is employed in" relation, and remove this relation, then the implementation will only consist of two record types as shown below



There is an alternative implementation shown [illegible] below using [illegible] set.

These two implementations will give the same possibilites in answering questions as the previous implementations and will require less space and time in use.

In general we can describe each object class by none, one, or more record types depending upon:

- the volume of data per object in the class
- how this volume varies for objects in the class
- the usage pattern and frequency of usage of the various data elements.

Correspondingly each relation class can be described by none, one, or more record types depending upon:

- the volume of data per relation in the class
- how this volume varies for relations in the same class
- the usage pattern and frequency of usage of the various data elements.
- the relation type (I:I, I:N, N:N)

When the choice of record types has been made, these should be documented in a "record type-data element table" as shown in the chapter on documentation.

4.5.6 Choice of Mechanisms

The above example shows how we can implement a conceptual model using different combinations of record types and access mechanisms. We have however, not discussed the choice of mechanisms. This will depend upon the way(s) in which the implementation is to be used.

To be able to choose mechanisms one must for each record type describe the following:

- How are the records in the record type to be retrieved, that is which are the relevant access keys. For each of these we must describe how often they will be used and how may records of the type (measured both absolutely and as a percentage of the total number of records in the type) are to be retrieved. Further, we must take account of the required response time and indicate whether the records are to be retrieved in a special orde

- How often new records are stored or old records deleted, and how many (measured absolutely and as a percentage of the total number of records in a record type).
  One must also know how often initial storing (loading) will occur.

- How often data elements are updated and which data elements are involved. Attention should especially be given to the updating of data elements which are also access keys.

- How many records are these of a given record type, both absolutely and as a percentage of the total number of records in the database. One should also have a good idea of how exact any estimates of these numbers are.

- How much of the total storage space required is to be occupied by each record type.

To obtain the figures necessary to describe the conditions listed above, each transaction to be run against the data base must be broken down to database operations per record type. In practice one may have to be satisfied with average values as the number of data base operations per transaction may be dependent upon the database's contents. The two tables give the characteristics of the various storage and access mechanisms.

The tables describe the characteristics and considerations only qualitatively. In practice a number of these must be made quantitative before a decision can be made .

By describing how each record type is to be used it is possible to decide the characteristics which the mechanisms should have, and use this as a basis for choosing a combination of mechanisms for each record type.

It can be seen that there is no one to one correspondence between usage patterns and mechanisms, but we feel it is important that one describes these systematically both to avoid leaving anything out and to give an awareness of possible alternatives.

Once the various mechanisms have been chosen, they should be documented via a "record type - set table" and a "data element table".

Storage mechanism Criteria	Initial Storage	Storing new records	Deletion of records	Effective use of space and 'holes"	Sensivity to changes in assumptions	Information Retrieval	Storage order
Random	Costly if no account of hashing mec-hansim is taken	Fast Execut-ing, existing records are not affected	Fast Execut-ing, existing records are not affected	Normally 70-90% usage of space. Holes can be used for new re-cords. .Space requirement independent of no. of re-cords in file	Sensitive to number of records if this greatly exceeds that originally expected. Distribution of key values is important	Serves also as access mechanism	Depends of value of access key and time of storage.
Sequential	Fast if records are presorted	Affects al-ready exist-ing records	Gives holes not easily reused	Up to 100%, Holes not reusable with-out reorgani-sation	Space utili-zation and re-sponse times strongly de-pendent upon storage and deletion of records	Well suited to sequential handling. Binary search techniques can be used	Depends on value of sort key.
Serial	Fast	Fast, does not affect records already in file	Fast. Does not affect other records in file	Up to 100%. Holes can usually be easily reused		Normally requires at least one access mechanism	Dependent solely on time of storage
Proximity	Fast	Fast	Fast	Not as good as seriel			Dependent upon re-cords logical relations. Or its content and storage instant

Characteristics of Storage Mechanisms

Access Mechanism / Criteria	Storage requirements	Retrieval of records	Storage of records	Deletion of records	Sensitivity to changes in original assumptions	Number of mechanisms possible per record
Random	None	Effective when fetching single records	Fast, no mechanism to be updated	Fast, no mechanism to be updated	Access time degraded if the number of records strongly exceeds that expected	One
Index table	Proportional to number of records	Look up time dependent upon no. of records	Tables must be updated	Tables must be updated		Many
Chaining	Proportional to number of records	Access time to individual records proportional to number of records in chain	Other records must be updated	Other records must be updated. Costly for long single chains		Many
Searching	None	Suitable when retrieving high proportion of records in file				Many

The Characteristics of Various Access Mechanisms.

5 AN EXAMPLE

The application of the construction rules is illustrated in the following rather larger example.

The example is based upon the Akergroups cost-accounting system Ag-kost. The system registers the costs of hours used, materials, EDP costs, the purchase of external labour or consulting in terms of hours, and the cost of materials purchased at the various cost points, which can be new buildings, repair orders, projects and so on.

The system is to be the basis for calculating and monitoring costs.

5.1 The Problem

The problem is described below, where objects are underlined with full lines, and relations with dotted lines.

An order (load point) can be divided into a number of jobs which may be carried out by one or more departments which then record hours on the jobs. An order can also be directly costed for hours by departments.

An order is identified by main characteristic/secondary characteristic. A job is identified by a work sheet number

Each department belongs to a factory.

Orders are also charged for use of stock material and direct costs. Direct costs can be the purchasing of external labour or direct purchasing of material, travelling expenses, EDP costs and so on.

Each usage of hours, stockmaterial or direct costs are given a cost type.

Orders are grouped into product groups like new building, repairs, and so on. Each order belongs to only one product group. And each order belongs to a factory.

The database to be designed must answer the following questions and will have the following transactions run against it:

1. Registration of the new usage of hours, stock material and direct costs.
2. The retrieval and updating of usage of hours, stock material and direct costs.
3. The retrieval and deletion of cost points (orders) with their corresponding use of stock materiel, time usage, and direct costs, when these are no longer of interest.
4. The retrieval of all cost points and corresponding usages for a given product group.
5. The retrieval of a given cost point and its corresponding resource usage.
6. The retrieval of cost points (orders) and their corresponding resource usage which are not yet settled, are settled, or have additional settlements.
7. The retrieval of the usage of stock material belonging to a given cost type.
8. The retrieval of cost points belonging to a given type of production.
9. The retrieval of all cost points for a given factory.

5.[illegible] The Conceptual Model

We can list the objects and relations which will make up the conceptual model. We choose to regard orders and jobs as objects rather than relations since they already have their own identification systems and because not all the objects which they would have been relations between are included in this situation. We also regard cost type and product group as object classes. We then have the following objects and relations.

Objects

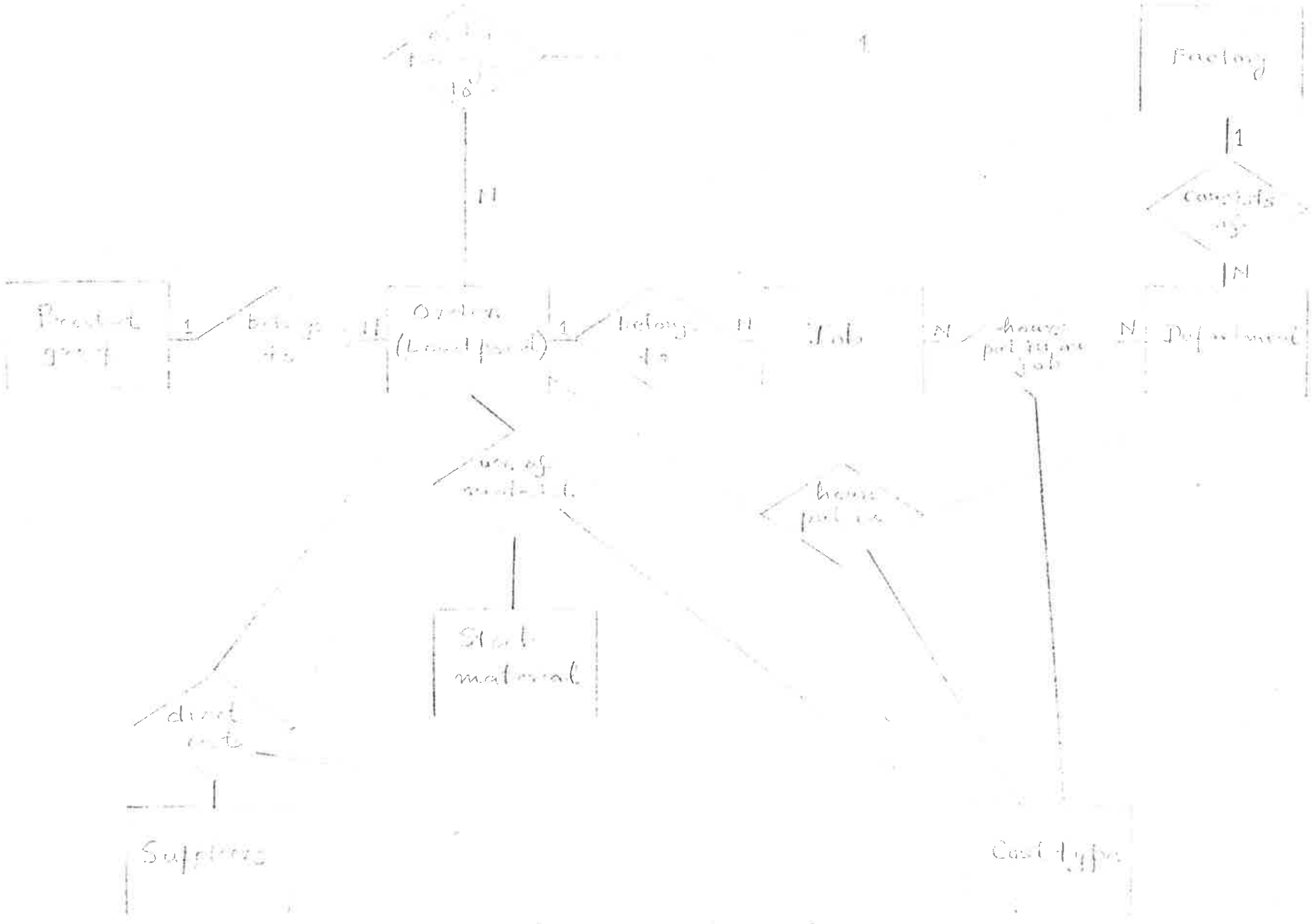
- Order
- Factory
- Department
- Job
- Stock material
- Cost type
- Suppliers
- Product group

Relations

- Factories consist of departments
- Departments put in hours of a given cost type on orders
- Departments put in hours of a given cost type on jobs
- Jobs belong to orders
- Orders use stock material of given cost types
- Direct supply of a given cost type to orders occurs from suppliers
- Orders belong to product groups
- Orders belong to factories

We can then draw the conceptual model shown opposite. Since an order can only belong to one factory or product group these relations will be 1:N. In addition a department belongs to only one factory, and a job belongs to only one order.

Using the transactions and inquiries which will eventually be run against the database, and the conceptual model, we can list the characteristics or proper ties of the various objects and relations.



The completed model.

Properity list for objects

Object	Identifying proper ties	Other proper ties
Factory	Factory number	
Department	Dept. number	
Job	Job number	
Order	Main access key Secondary access key	Settlement status Date settelsed
Stock material	Material identification	
Cost type	Cost type number	
Supplier	Supplier code	
Product group	Product group number	
Relation		
Order belongs to	Main key Secondary key Factory number	
Consists of	Factory number Department number	
	Product group number Main key Secondary key	
Job belongs to	Main key Secondary key Job number	
Hours put in an job	Job number, Department number	Number of hours, period over which hours wer put in
	Cost type	Sum
Hours put in	Main key Secondary key Dept number Cost type	Number of hours, period sum
Stock used	Main key Secondary key Material i.d. Cost type	Amount, units code period, sum
Direct costs	Main key Secondary key Supplier code Cost type	Amount, units code period, sum

The various transactions or the answering of queries are carried out against the model as follows:

1 The recording of new relations

hours_worked_on job
hours_worked (direct)
material use
direct_costs

2 Access of relations

hours worked
stock used
direct costs

via order. Access of relation hours worked on_job via order, job_belongs to and job.

3. Searching through the whole object class order

4 Access of orders via product group and relation consists of

5 Retrieval of order using main access key or secondary access key. Retrieval of relations hours worked, material used, and direct_costs. Access of relation hours worked_on job via job belongs_to and job.

6 Searching through whole object class order. Relevant use of resources fetched as in 5.

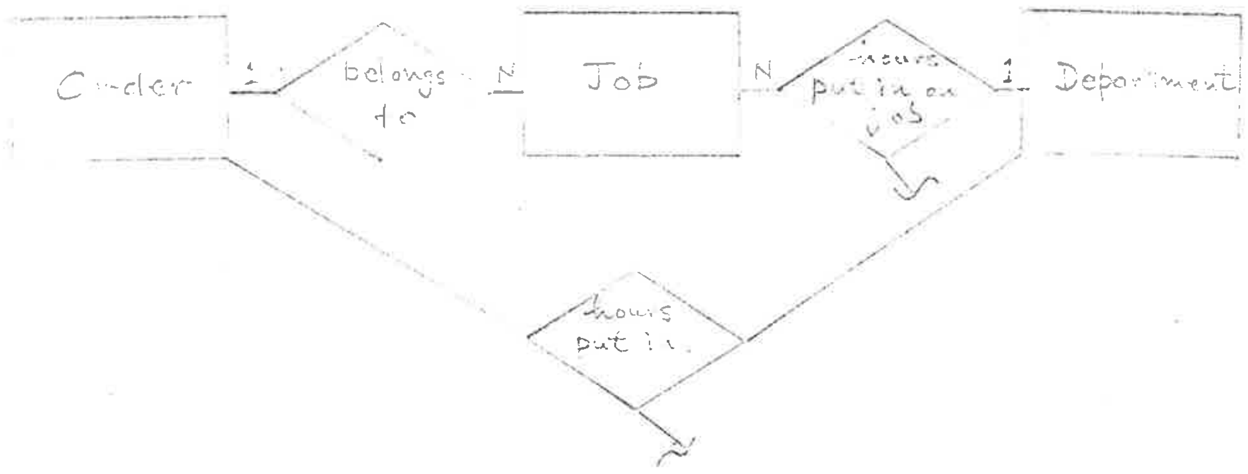
7 Retrieval of material used via cost type

8 Access of orders via product group and the relation belongs to

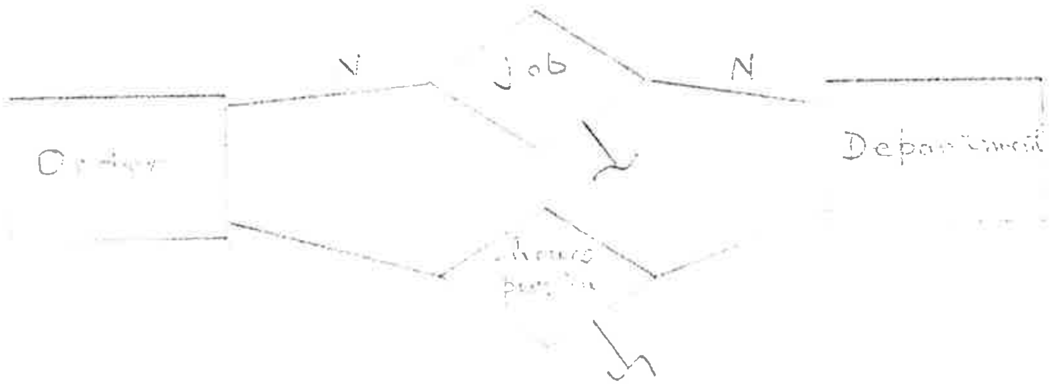
9 Access of orders via factory and the relation belongs to

5.2.1 Reduction of the Conceptual Model

If we consider the conceptual model shown earlier , and look for the worthwhile reductions. we can look especially at that part of it reproduced below

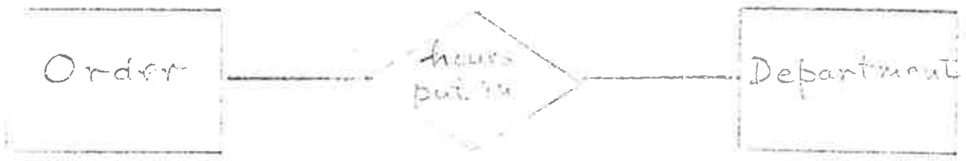


If we put the object class Job and the two surrounding relations together we get an N:N relation called Job as indicated below (see the construction rules: reduction of two relations with an object in between to one relation)

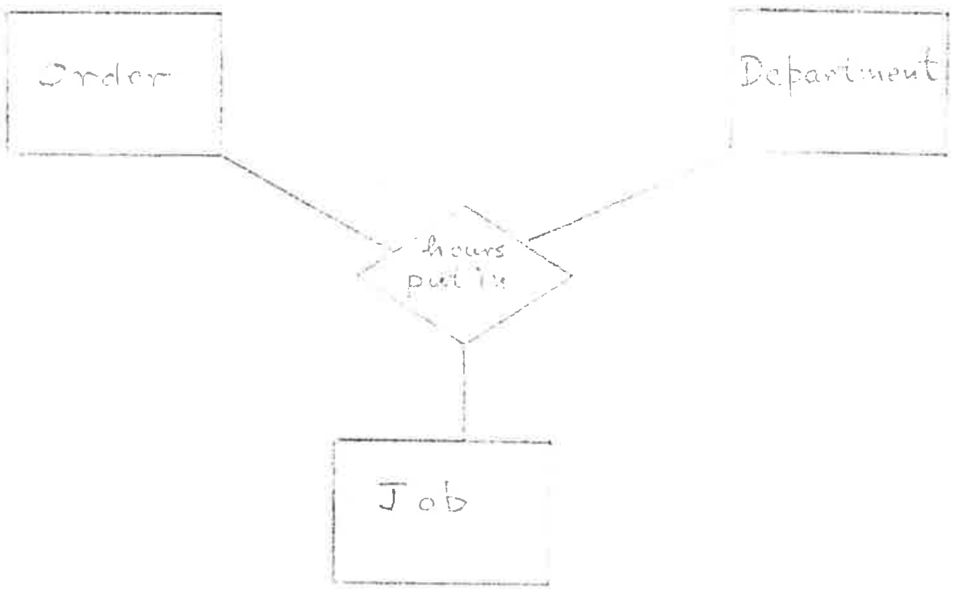


We now have two relations, hours worked (direct), and job, which in fact describe the same thing, namely that departments work on an order and charge it for hours worked. The difference between them is in the details of the work done. For the relation "hours_worked_on" only the relevant order is specified, but the work done on a job is specified via the job type and the hours worked.

If Agkost requires jobs to be specified in any detail, then the information will lie in the relation "job". If this is not the case then the two relations can be combined into one. For the hours worked directly on an order, the relation will be over-dimensioned as no job number is defined. The combined relations will then be as shown below:

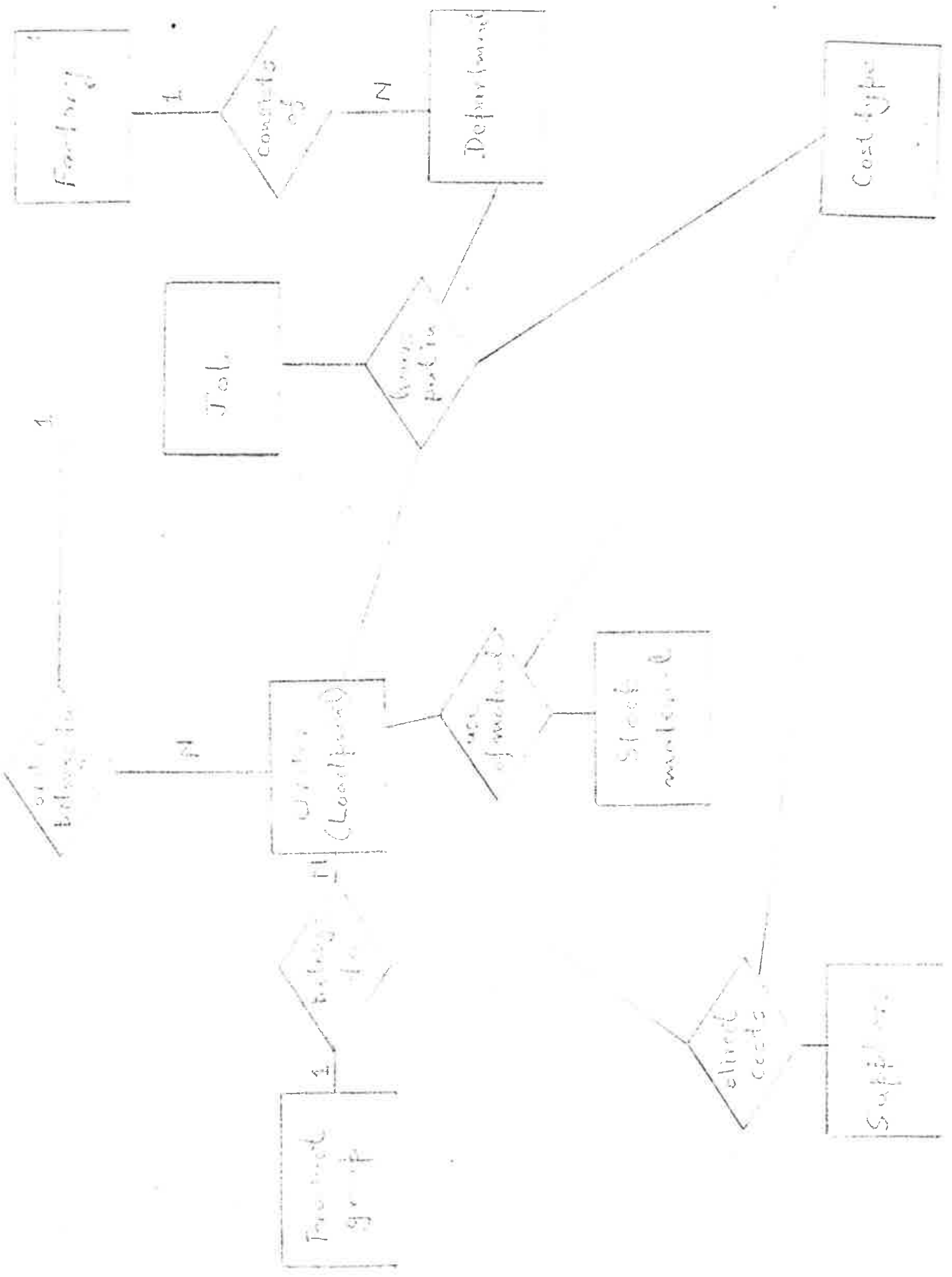


The part of the model shown at the top of the previous page can be reduced in another way. The model consists of 3 object classes joined by 3 "binary" relations. It can therefore be transformed to three objects joined via one third order relation, as shown below (see the construction rules).



As opposed to the model above it this last model has the object class job. If the model is to contain information about jobs, other than job number, then this will belong in the object class job. The content of the relation "hours put in" will be the same for both the previous models.

Using the result last [illegible] the whole conceptual model will appear as shown below:



The reduced conceptual model.

5.2.3 Representation in Terms of Record Types

Using the conceptual model shown above, the various transactions and queries which are tested against the model. and the properties necessary to describe the various object and relation types. we can now decide what record types we need to implement the conceptual model.

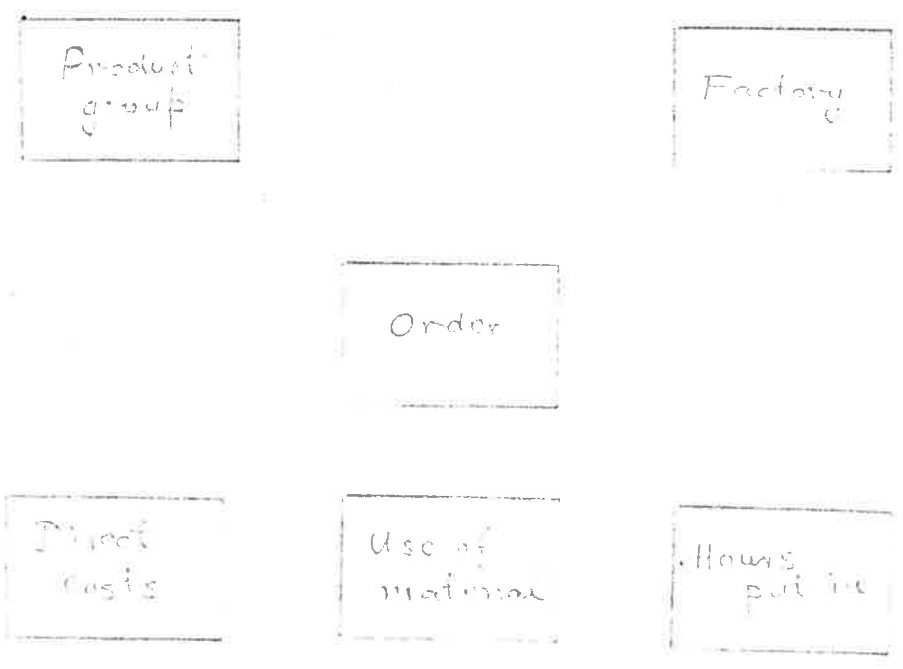
Only the identification is required for the object classes supplier, stock material, department and job. The relations connected to these object classes are never accessed via the object classes, thus they need not be implemented as record types.

Only the identification is required for the object class cost type. The relations in the class material use are to be fetched via cost type. However, this transaction is run so seldom that instead of keeping cost tupe as a record type we will access the material type records via order records even though we will have to retrieve a number of these material use records unnecessarily.

As the relations "belong to" and "order belongs to" are of the type 1:N we can move the content of the relations over to the order side (the "N" side) without introducing any need for variable record length. Thus the order records will then contain both factory number and product group number in addition to the other information describing the order.

As the object class "department" is not to be implemented via a record type, and its relation to factory is not required We have now got the following record types.

Product group, Factory, Order, Hours worked, Material usage and Direct costs.



It may be relevant to add some record types beyond those we have so far obtained to increase the processing effectivity of the system. The transactions and queries to be run against the data base involve summations in a number of cases which requires a level of detail above that of the order records. Dependent upon the frequency of updating or retrieval, it may be relevant ot store these required sums in one or more record types. We will return to this point when calculating the database loading from these frequencies. The record types and their data elements are shown in the table opposite.
We can see that the record types production type and factory at present only contain an identification. We choose however to retain them as record types since product type and factory number are used to access order records. It may also be useful to store some of the agregated quantities in these record types.

Record Type, Name / Data Element Name	Product Group	Factory	Order (lead point)	Hours put in	Use of Material	Direct Costs				
Product Group	X		X	X	X	X				
Factory Number		X	X	X	X	X				
Main Access Key			X	X	X	X				
Secondary Access key			X	X	X	X				
Code			X							
Settlement Status			X	X	X	X				
Date Settled			X	X	X	X				
Agregates, hours,cost			X							
Dept. Number				X	X	X				
Cost Type				X	X	X				
Job Number				X	X	X				
Detail Code				X	X	X				
Load Period				X	X	X				
Amount				X	X	X				
Unit Code				X	X	X				
Cost				X	X	X				
Order Number				X	X	X				
Account Number				X	X	X				

Record Type/Data Element Table

Record Type:	Operation:	No. of Records:	Access Key:	No. of Transfers per Year:	Total No. of Records per Year:
Detail Record	Retrieve	1000	Unique ID 1)	52	52000
Detail Record	Update	1000	---	52	52000
Detail Record	Scan	20000	Unique ID	52	1040000
Detail Record	Update	5000	---	52	260000
Detail Record	Store	15000	---	52	780000
Detail Record	Fetch	2500	Main key, secondary key	52	130000
Detail Record	Fetch	80000	Main key, secondary key, date of settlement	12	960000
Detail Record	Delete	80000	---	12	960000
Detail Record	Fetch	10000	Main key, secondary key, date of settlement	12	120000
Detail Record	Fetch	3000	Cost type	12	36000
Detail Record	Fetch	180000	Product group, main key, secondary key	1	180000
Detail Record	Delete	180000	---	1	180000
Detail Record	Fetch	2000	Main key, secondary key, settlement status	1	2000
Detail Record	Delete	2000	---	1	2000
Detail Record	Fetch	2000	Main key, secondary key, settlement status	1	2000

continued on following page

Record Type:	Operation:	No. of Records:	Access Key:	No. of Transfers per Year:	Total No. of Records per Year:
Order Record	Fetch	2500	Main key, secondary key	52	130000
Order Record	Update	2500	—	52	130000
Order Record	Fetch	250	Main key, secondary key	52	13000
Order Record	Update	250	—	52	13000
Order Record	Fetch	50	Main key, secondary key	52	2600
Order Record	Fetch	10000	Product group	52	520000
Order Record	Fetch	100	Product group	52	5200
Order Record	Fetch	800	Date of settlement	12	9600
Order Record	Fetch	100	Date of settlement	12	2400
Order Record	Fetch	200	Date of settlement	12	2400
Order Record	Fetch	1000	Product group	12	12000
Order Record	Fetch	1000	Product group	12	12000
Order Record	Fetch	20	Main key, secondary key, settlement status	1	20
Order Record	Fetch	5000	Main key, secondary key	1	5000

5.3.1 Retrieval, Storage, Updating and Deletion of Records

From the various record types we now have, and the transactions/queries to be run against the database we can list how, how often, and how many of the various record types must be handled in a given time (stored, retrieved, updated or deleted). We use the following figures which are estimates for NV. (Nylands Verksted in Oslo)

Record type	Number
Production type	1
Factory	9
Order	14000
Timed worked	630000
Material use	255000
Direct cost	100000

Hereafter the record types time worked, material use and direct costs will be lumped together under the term "detail record".

To calculate the load on the database we use the following procedure:

- Each transaction and question is broken down into database operations per record type. Account is taken of the fact that update or deletion of records also involves retrieval.

- For retrieval the access key or search mechanism is given.

This break down is not covered here in detail, but only the results given in the table opposite.

The unique access key (or identification) for the detail records is Factory, main key and secondary key, cost type, department number, job number detail code. (all together)

The table above does not take account of the storing and deletion of order records. The number of order records is about 1% of the number of detail records and this is less than the uncertainty involved in the complete calculation and can be ignored.

If we add up the number records retrieved using a given access key per year, we obtain the figures given in the table below

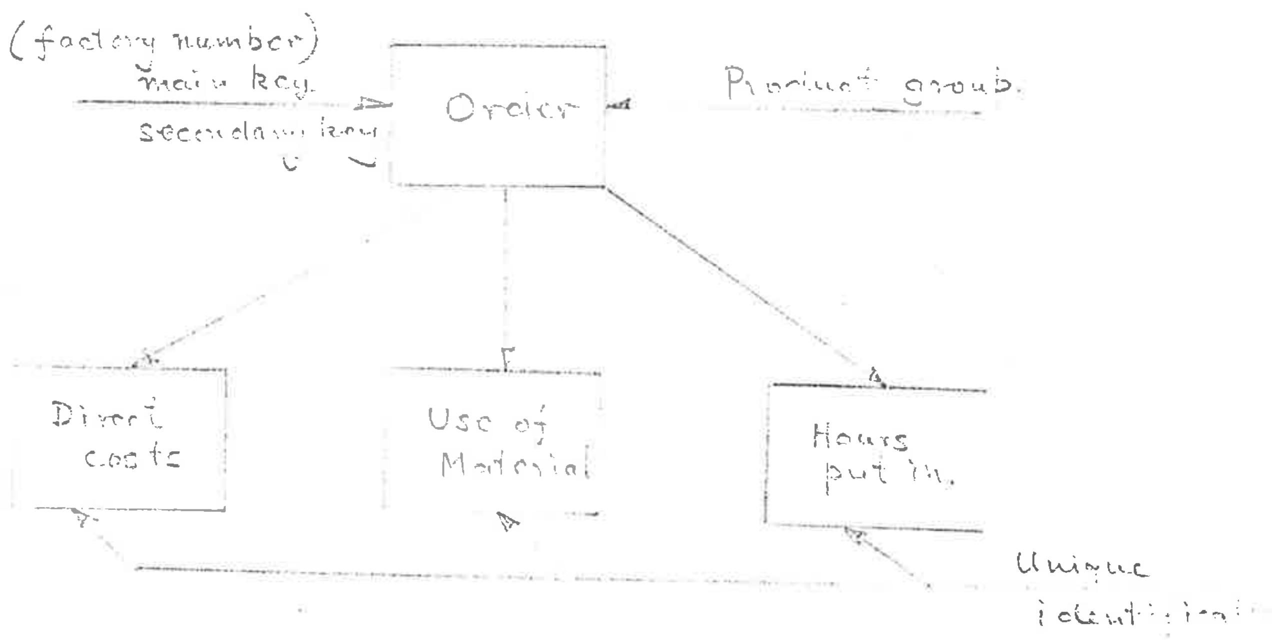
Record Type:	Access Key:	No. of Records/Year:	No. of Runs per Year:	Records/Run:
Detail Record	Unique ID	1092000	104	10000
Detail Record	Main key, secondary key	130000	52	2500
Detail Record	Main key, secondary key, Data of. settlement	1080000	24	45000
Detail Record	Main key, secondary cost type	36000	12	3000
Detail Record	Main key, secondary settlement status	4000	2	2000
Detail Record	product art., main key, secondary key	180000	1	180000
Order Record	Main key, secondary key	150600	157	950
Order Record	Product group	549200	128	4400
Order Record	Date of settlement	14400	36	400
Order Record	Main key, secondary key, Settlement status	20	1	20
Order Record	Main key	5000	1	5000

5.3.2 Choice of Mechanisms

As can be seen all retrieval goes via Main access key, secondary access key, so that we have a purely hierarchic structure. This indicates that a sequential organisation sorted upon the whole of the unique identification could be advantageous since the file would never have to be resorted. However, as only a small fraction of the total number of records are accessed in each transaction program it is more advantageous to choose a solution using various access mechanisms.

Here we work through the example and satisfy the various processing requirements using various access mechanisms.

The required access paths lead us to the implementation structure shown below:



For the order records we have choosen access via an index on factory number, main key, secodary key combined in that order. This access key is hierarchic and we can for example fetch all records with the same factory number, all records with the same factory number and main key, and any single record with a combination of factory number, main key and secondary key.

We have also chosen to have access via production type so that one can access all records with the same production type.

The order records and the detail records are chained together in sets such that one can access all the detail records belonging to a given order.

In addition, the detail posts have access via hashing on the unique identification (that is the combination facotry number, main key, secondary key, cost type, department number, job number and detail code).

With these mechanisms we satisfy all the access paths except those based on time and status and cost type.

If we know the resource requirements (CPU time, channel time, and number of accesses) for each database operation as a function of mechanism type we can calculate the annual resource requirements and costs.

In the implementation structure shown in fig. 62 access via hashing upon the unique identification for detail records has been chosen. This means that the files to contain detail records must be dimensioned to hold the maximum number of records expected since expansion of a hashing file is costly.

This can be avoided by organising the detail records as serial files which can more easily be expanded. However, we then lose the direct access via unique identification and must go via the set which connects detail records to the order records. This will result in the transactions using the unique id taking longer time.

As can be seen the retrieval, storage or deletion of detail records takes most of the time. If we instead of having single detail records made records with space for 2. 3. 4... detail records by sorting usage with same department number or cost type, the total number of records will decrease correspondingly. The total space requirements will also be lessened since the same main key , secondary key will not be repeated in so many records. The space required by pointers will also decrease.

How much we can achieve in this way depends upon how many detail records with the same cost type or department number are to be found at each load point.

REFERENCES

Kjeld Koushede

Databaser, begreper, teknikk og konstruksjon
FORUM 1973

Per Reinholdt, Thomas Skousen

Konstruksjon av intergrerte systemer
FORUM 1972

Nils Fredholm

Databasemetodikk
Studentlitteratur 1972

Hofstad Wibe Bratsbergengen

Filsystemer og Databaser
Tapir 1973

L. M. Iversland, L. B. Metlie, A. Nilsen

Data structure matrix

Mats Lundberg, Erling S. Andersen

Systemering – Informasjonsanalyse

Codasyl Data Base Task Group

April 1971 – report

Database Systems: A Practical Reference
Ian Palmer