Base Operators as a Tool for Systems Development

Bo Sundgren

SCB

R & D Report
Statistics Sweden
Research - Methods - Development
1988:3

INLEDNING

TILL

R & D report : research, methods, development / Statistics Sweden. – Stockholm : Statistiska centralbyrån, 1988-2004. – Nr. 1988:1-2004:2.
Häri ingår Abstracts : sammanfattningar av metodrapporter från SCB med egen numrering.

Föregångare:

Metodinformation : preliminär rapport från Statistiska centralbyrån. – Stockholm : Statistiska centralbyrån. – 1984-1986. – Nr 1984:1-1986:8.

U/ADB / Statistics Sweden. – Stockholm : Statistiska centralbyrån, 1986-1987. – Nr E24-E26

R & D report : research, methods, development, U/STM / Statistics Sweden. – Stockholm : Statistiska centralbyrån, 1987. – Nr 29-41.

Efterföljare:

Research and development : methodology reports from Statistics Sweden. – Stockholm : Statistiska centralbyrån. – 2006-. – Nr 2006:1-.

R & D Report 1988:3. Base Operators as a tool for systems development / Bo Sundgren.
Digitaliserad av Statistiska centralbyrån (SCB) 2016.

urn:nbn:se:scb-1988-X101OP8803

Base Operators as a Tool for Systems Development

Bo Sundgren

R & D Report
Statistics Sweden
Research - Methods - Development
1988:3

Published June, 1988
Printer STATISTICS SWEDEN, Dept of Research and Development, EDP Systems Unit
Publisher Staffan Wahlström
Questions answered by Bo Sundgren

ISSN 0283-8680

Printed in Sweden
Garnisonstryckeriet, Stockholm 1988

STATISTICS SWEDEN
Bo Sundgren

1988-04-26

BASE OPERATORS AS A TOOL FOR SYSTEMS DEVELOPMENT

Schematically, the design of a system for the processing of the data that have been collected in a statistical survey may be structured into three steps:

1. design of the input data structures

2. design of the output data structures

3. design of the input-->output transformation process structure

Ideally, step 2 should precede step 1, since the input data and input data structures should be determined by the output data and output data structures demanded by the users of the statistical survey. However, in practice it is not uncommon that one has already designed the questionnaire for collecting the input data and even carried out the full scale data collection, before one really starts thinking about the output in terms of tabulation plans, etc.

1 Design of the input data structures

The design of input data structures proceeds in two steps:

1. design of an infological object system model for the input data

2. transformation of the object system model into a flat file structure

1.1 Design of an object system model

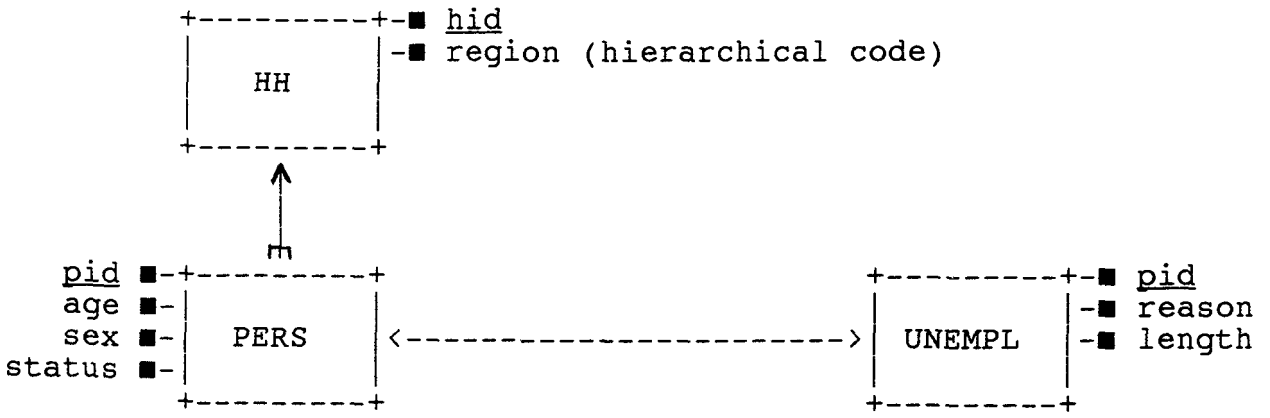
Given an existing questionnaire, and maybe even an existing data collection, the design of an object system model will become a kind of reconstruction of something that should ideally have been constructed **before** the design and construction of the questionnaire. Thus the object system model will be based upon

an a retroactive analysis of the input data and the forms for collecting the input data.

On the other hand, if a proper infological analysis has taken place **before** the design and construction of the questionnaire, we will use the object system model that has already been developed, with possible modifications, reflecting later changes in the design.

Example: A Household Survey

In a household oriented survey, like a labour force survey, there are two obvious objects: **households, HH,** and individual **persons, PERS.** In addition, there may be important subcategories, like the sub-object type of **unemployed** persons, **UNEMPL.** There will be a total one-to-many relationship between households and persons, and a partial one-to-one relationship between persons and unemployed persons. For each one of the objects there will be a set of variables, like HH: id, region; PERS: id, age, sex, status_of_the_person_within_the_household; UNEMPL: id, reason_for_unemployment, length_of_unemployment. This object system model may be visualized in the following way:



1.2 Design of the file structure

Any object system model may be transformed into a structure of flat files by means of the following transformation rules:

Rule 1: object type --> flat file;

Rule 2: many-to-many object relation --> flat file;

Rule 3: many-to-one object relation --> foreign key;

In the household survey example we will get:

HH --> HH(hid, region) according to Rule 1;

PERS --> PERS(pid, age, sex, status, hid■) according to Rule 1&3;

UNEMPL --> UNEMPL(pid■, reason, length) according to Rule 1&3;

Note: primary key, foreign key ■;

2 Design of the output data structures

A tabulation plan may be a good starting point for the design of the output-oriented file structure of a statistical data processing system. However, such a listing of tables to be produced does not usually possess the necessary degree of precision and unambiguity. For example, have a look at the following tabulation request:

"The population of Zimbabwe by region, size of household, and by age and sex of the head of household."

So-called alfa-beta-gamma-($\alpha\beta\gamma$)-analysis may be used for clarify-ing the different interpretation alternatives of such a request, and for making a final decision about precisely which output is actually demanded by the user who has made the request.

2.1 Alfa-beta-gamma-analysis of the demanded output

An alfa-beta-gamma-analysis follows the pattern:

α: for <object type> with <property>

β: give <list of statistical variables>

γ: by <list of variables>

In our example we may get:

α: for PERS with

ß: give count

γ: by region, size_of_hh, agegroup_of_head, sex_of_head

Note: denotes any property held by all objects in <object type>

2.2 Analysis of derived variables

If we compare the variables appearing in the alfa-beta-gamma-analysis of the demanded output with the variables in the input-based model of the object system, we will find that some of the former have to be derived, and fortunately can be derived, from the latter. Thus:

- agegroup = g(age); where g denotes a function that classifies, or groups, the values of a variable
- region(PERS) = HH.region;
- size_of_hh(HH) = PERS.count;
- size_of_hh(PERS) = HH.size_of_hh = HH.PERS.count;
- HEAD(HH) = PERS(with status = 1); **note:** derivation of object
- sex_of_head(HH) = sex(HEAD(HH)) =
  = PERS(with status = 1).sex;
- sex_of_head(PERS) = HH.PERS(with status = 1).sex;
- agegroup_of_head(PERS) = HH.PERS(with status =1). g(age);

The definitions of derived variables may be included as a fourth component, delta (δ), in the alfa-beta-gamma-pattern. For example, in our example we may state:

α: for PERS with

ß: give count

γ: by region, size_of_hh, agegroup_of_head, sex_of_head

δ: where region = HH.region,

```
        size_of_hh = HH.PERS.count,
        sex_of_head = HH.PERS(with status = 1).sex,
        agegroup_of_head = HH.PERS(with status = 1).
                           g(age)
```

or alternatively:

```
 : for PERS with

ß: give count

 : by region, size_of_hh, agegroup_of_head, sex_of_head

 : where region = HH.region,
         size_of_hh = HH.PERS.count,
         sex_of_head = HEAD.sex,
         agegroup_of_head = HEAD.g(age),
         HEAD = HH.PERS(with status = 1)
```

2.3 Finding the target file

Having done the alfa-beta-gamma-analysis we may relative easily specify the **target file**, that is, the file from which a particular aggregation/tabulation can easily be made. Like the input files, the target file should be a flat file. In our case it is obvious that the most suitable flat file for an aggregation resulting in the demanded result would be a PERS file, or more precisely the following one:

```
PERS0 = PERS0(region, size_of_hh, agegroup_of_head,
        sex_of_head)
```

where the derived variables are defined as above. Note that the target file need not necessarily contain an identifier. Thus duplicates may appear, and should be counted as different objects.

3 Design of the input-->output transformation

In some situations the target file will simply be one of the flat files in the input structure. Then no input-->output transformation at all will be needed. We just have to specify the appropriate aggregation/-tabulation.

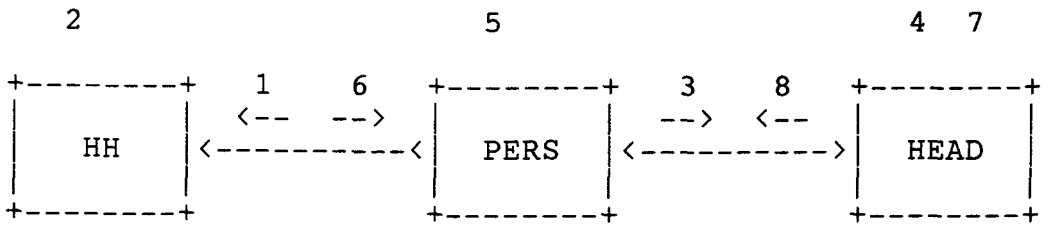
However, in most cases some kind of non-trivial transformations between the input-oriented and the output-oriented file structures have to take place. The base operator approach will then offer a systematical way

of finding and describing the logically necessary steps. The actual implementation of the transformation steps may be in terms of the existing BOS, the base operator package developed by the UN/ECE Statistical Computing Project (SCP), but other popular software products for data manipulation, like SAS and EASY-TRIEVE, may also be used.

In our example the following sequence of base operators will do the job:

1. aggregate PERS by hid count = size_of_hh giving HH1;
2. join HH, HH1 where hid = hid giving HH2;
3. select PERS where status = 1 giving HEADS;
4. define HEADS set agegroup = ... giving HEADS1;
5. project PERS over hid giving PERS1;
6. join HH2, PERS1 where hid = hid giving PERS2;
7. project HEADS1 over hid, sex, agegroup giving HEADS2;
8. join HEADS2, PERS2 where hid = hid giving PERS0;
9. aggregate PERS0 by region, size_of_hh, agegroup, sex giving TAB;

The chain of base operators may also be represented as a flow of movements in the visualization of the object system model:



4 Some improvements of the proposed methodology

The solution presented above is relatively straight-forward and easy to understand, although the requested table is a rather complex one. The solution also has the advantage that it can be implemented at once, since

it complies 100% with the syntax of the existing software - the Base Operator System (BOS) developed by the Joint Group on Statistical Data Base Management within the UN/ECE Statistical Computing Project.

However, a critical observer may make some objections to the solution presented above:

1. It consists of a large number of steps, and it is not obvious how and why one has arrived at this particular solution rather than some logically equivalent solution that would also solve the problem.

2. The process of solving the problems seems to be unstructured in the sense that one goes from the problem specification to the detailed solution in one step.

3. It may be easy to make small errors when writing the base operator statements.

Here are some proposals for improving these matters:

1. Before writing a sequence of statements on the base operator level, make an explicit outline of the strategy to be followed in the transformation process. The strategy should consist of a small number of major steps, each major step should be broken down into (sub)steps, etc, until one reaches the level of individual base operators.

2. Even though the Base Operator System automatically produces the record descriptions of the output files, it could be practical, in order to avoid errors during the design process, to state explicitly (and redundantly) the columns of the output relations.

3. Projections are usually rather trivial operations with the aim of reducing unnecessarily large volumes of data. By "infixing" them as substatements within other base operator statements, or by implying them by explicitly leaving out certain columns in the output descriptions mentioned in the previous proposal, one could reduce the number of steps, and put more concentrated attention on the more important steps in the transformation.

If we take these proposals into account, an alternative solution to our example problem might be as follows.

Transformation strategy:

A. Create the household-related variables, and adjoin

them to the persons.

B. Create the head_of_household-related variables and adjoin them to the persons.

C. Aggregate the persons.

Specification of Step A:

```
A1. aggregate PERS by hid count = size_of_hh
          giving HH1(hid, size_of_hh);

A2. join HH, HH1 where hid = hid
          giving HH2(hid, region, size_of_hh);

A3. join HH2, PERS(pid, hid■) where hid = hid
          giving PERS1(pid, hid■, region, size_of_hh);
```

Note. Substep A3 contains an "infixed" projection of PERS.

Specification of Step B:

```
B1. select PERS where status = 1
          giving HEADS(pid■, age, sex, status = 1,
                       hid■);

B2. define HEADS set agegroup = ...
          giving HEADS1(hid■, agegroup, sex);

B3. join HEADS1, PERS1 where hid = hid
          giving PERS0(pid, region, size_of_hh,
                       agegroup = agegroup_of_head,
                       sex = sex_of_head);
```

Note 1. Substeps B2 and B3 contain implied projections.

Note 2. In substep B1 we make a conceptual shift of the primary key from pid to hid. This reflects our intention to adjoin, in substep B3, the data of heads_of_households to the data about person **via the common hid column**, rather than via the common pid column. Actually, this corresponds conceptually to a two-step join **via households.** An alternative, and maybe more stringent way of modelling this would be possible if the base operator set contained a more generalized **aggregation** operator, which would move selected heads_of_household data from the (sorted) person file to an (aggregated) household file. It could look like this:

```
"aggregate PERS by hid retrieve age, sex
          where status = 1 giving HH3(hid, age, sex)"
```

The typical characteristic of such a generalized agg-

regation operator would be that it would always produce, from each successive **group of rows** identified by the "by clause", **one single row** in the output relation. Thus in our case it must be assumed that there is only one member of each household that has "status = 1". Otherwize the operator would have to be defined so as to retrieve the requested data from one of the "competing" heads_of_house-hold, probably the first one located.

Note 3. In substep B3 we have renamed some columns in the output. This syntax cannot be used in the present implementation of the base operators, but there is a special rename base operator. In order to make the systems design easy to understand, it is important to choose informative names of columns, and to change them appropriately when they are adjoined to other relation, if it is necessary to avoid confusion and misinterpretation.

Note 4. In substep B3 the pid column could have been projected away from the output, since it is will not be needed in the subsequent aggregation process in step C. We have kept it mainly for the sake of clarity, to remind us that it is person records that are going to be counted.

Specification of Step C:

```
C1. aggregate PERS0 by region, size_of_hh,
              agegroup_of_head, sex_of_head
              count = population
          giving TAB(region, size_of_hh,
agegroup_of_head, sex_of_head,
              population);
```

Note. The gamma-variables of the request become the primary key of the aggregated file. The final editing and layout of the actual table to be presented to the user is, at least at present, outside the scope of the base operator approach.

5 Possible usage of the base operator approach in the development of generalized software

So far we have discussed the possibilities of using the base operator approach as a tool in the development of **application systems.** However, one could also consider the usage of the base operator approach in the development of **generalized software.** This has been discussed from time to time within the framework of the UN/ECE Statistical Computing Project (SCP). Naturally, these discussions started within the Joint Group on Statistical Data Base Management, since it was within this

group that the design and development of an actual piece of software, the Base Operator System (BOS), based on the base operator philosophy, was initiated and carried out. BOS is an extension of the relational algebra for statistical purposes. Like the relational algebra it primarily aims at covering basic data manipulation operations, and it could serve as the data base management component of a generalized software system for the processing of statistical surveys.

The member countries of the Joint Group on Statistical Data Base Management regarded the development of the Base Operator System as a great success, not least as a contribution to the difficult problem of how to design generalized software. Thus besides developing a useful piece of software, the Joint Group also developed what seemed to be an interesting and successful design technique for generalized software development in general. Since several of the member countries of the Joint Group on Statistical Data Base Management also happened to be members of other Joint Groups of the SCP, which were also developing generalized software, the question was asked whether the seemingly successful base operator approach could not be generalized and applied also in the work of these other Joint Groups - notably in the Data Editing Joint Group, and in the Joint Group on Tabulation (INTERTAB). After some rounds of discussions the answer seems to be a unanimous "yes, probably" and "let's try it".

What would be the advantages of using the base operator approach more widely in the development of generalized software for the processing of statistical surveys? One major advantage is that the base operator approach leads to a natural breakdown of any proposed piece of software into elementary, well-defined functions. For example, each base operator in the now existing BOS corresponds to an elementary, well-defined function for data manipulation. Since data manipulation functions are somehow parts of all systems for the processing of statistical surveys, the likelihood is very high that these operators, which have already been designed, developed, and implemented, could also be used as components in other packages, also in software systems whose primary aim is something else than pure and simple data manipulation. Such a discovery may save a lot of reinventions of wheels.

It is also likely that other functions in a statistical data processing system than pure data manipulation could also be designed and developed in analogy with the base operators in BOS. One important aspect of the base operator approach is that the base operators form an **algebra**: they use and produce entities of the same kind, relations and/or flat files, which means that they can be combined arbitrarily. Thus on the basis of

only a small set of well chosen base operators, one may form an endless number of different software systems, for a large variety of purposes.

A consequence of the modularity, based on functionality, and the combinability implied by the base operator approach, is that the development of new software products will be simpler and less expensive. Since the basic design strategy and methodology will already exist, and be well established, the systems design and the planning of the programming activities will be much simplified, and since many modules will already exist, and can be taken "off the shelf", the actual construction (that is programming) work will also be less resource-consuming. The software products will be easier to maintain, since they will to a large extent consist of the same, standardized components. This should also lead to improvements in software quality.

But what happens if a designer of a new software product is not satisfied with some existing base operator? Maybe the technical efficiency is not sufficient for the new purposes. Such a discovery may lead to the decision that a new version of the base operator has to be developed. When this development has taken place, the improvement will automatically be made available to all other software products using this function. Because of the modularity, based on functionality, and combinability of the components, a replacement of one component will not affect the others.

The benefits of the base operator approach require strict adherence to some simple but important design principles and standards. However, if one sticks to these common principles, one will also achieve a dramatical increase in the **integratability** of different software products. The principles mentioned above will ensure that the resulting software products will have a truly **open architecture**, a characteristic which is unforunately very uncommon in contemporary commercial software products. Today's software packages are still very **monolitic**. They usually do not lend themselves to easy and effiecient integration with other software products, an understandable consequence of the hard competition in the commercial world. Unfortunately, in this particular case, the competition does not seem tolead to the best result from the users' point of view.

Talking about the interests of the users, one may ask whether the base operator approach does not lead to too much standardization. However, sometimes standardization on some simple, basic principles, some elementary components, and some internal interfaces, in the end seems to result in a higher degree of flexibility on the level which is of primary interest to the user. We

have already discussed how it could be a rather limited and easy task to optimize some component or other, if this turn out to be essential for the total efficiency of the system. If different categories of users need different versions of some component, the different version could easily be shifted in and out without affecting the other components or the architecture of the system as a whole.

One of the most important concerns of the users is of course the **end-user interface**. Here again one may ask whether the base operator approach will not put too much of a strait-jacket on the designer of a particular software product. It is true, of course, that the base operators must follow a rather strictly defined syntax. However, there may be variations even within this strict framework, and, maybe more importantly, due to the **openness** of the architecture, it will be very easy to combine the standardized base operator approach with highly tailor-made user interfaces, or interfaces complying to interfaces/syntaxes which are already well-known to the users in a particular organization.

6 An example: generalized software for data editing

With a software product for data editing one should first of all be able to make different types of checking of the data:

- validity checks
- logical checks / consistency checks
- plausibility checks

When the checking leads to detection of possible errors, the data could either be automatically corrected, **imputed**, or they could be listed or displayed to a person, whether or not, and if so, how, the data should be changed. The correction could take place interactively or in batch mode. In both cases the corrected data should normally be rechecked, which may lead to another round of correction, etc.

It is relatively easy to see how some important functions of the data editing process could be expressed in terms of already existing base operators or minor generalizations of existing base operators. For example, many checking operations can be expressed in terms of define and select operators. The define operator can be used for setting a Boolean variable to "true" or "false" (or 1 or 0), depending on a condition that is expressed in terms of the variables represented in the file under consideration. Since the editing of a statistical file often involves a great number of checks, it may be practical to have a generalized define that

we may call multidefine where we could specify all the checks at once. Such an operator is already available in the present BOS in the shape of a **macro**. One could even go one step further and define a special edit-define that would not require explicit mentioning of the Boolean variables; it would be sufficient just to mention the error conditions (or correctness conditions, if one wants the checks to be formulated in that way). Each condition would have to have a name, though, so that it can be referred to later in the process.

Rules for automatic correction, or imputation, could also be specified with the define base operator. However, in this case the define operator should be defined in such a way as to give a new value to an existing variable, rather than creating a new one. Maybe we could call this a redefine operation.

If a human is involved in the correction process, either in batch mode or interactively, the variables to be corrected will be redefined on the basis of explicit values rather than expressions, as in the case of automatic correction.

The operations that we have mentioned so far are, in principle, all that is necessary for the editing process, as long as we are working with a single flat file or relation. In a statistical office maybe 90% of all editing is of this character. However, there are also some important, more complex situations. The most important one has to do with so-called **checking between files**. For example, let us assume that we have carried out a household survey. Some of the collected data will then concern the household as a whole, whereas others will concern individual members of the household. When put into a relational data base, such data will be split into two files or more: at least one file about households, and at least one file about persons; cf the example which was used in earlier sections of this paper. Many of the checks to be made during the editing of this data base will also be related **either** to households **or** to persons. Such checks can be handled as discussed above. However, there may also be checks that relate to a combination of household data and person data, for example so-called **structural checks**. This is an example of "checking between files". The files involved in such checking are usually hierarchically related to each other, on two or more levels, but more complex "many-to-many" situations are also possible, at least theoretically.

In order to be able to carry out "checking between files" one must obviously somehow bring together data from two or more files. This can always be done by means of the join base operator, which is applied one or more times, until one has obtained a **target file**

that contains all the data necessary for the editing operations (cf the discussion of target files earlier in this paper). But all the problems are not solved yet. Very often it is not sufficient to inspect the target file **row by row.** If the target file has an inherent hierarchical structure (like in the household/person example), it may be necessary to look at the file **partition by partition,** where a partion is defined by a **partitioning key,** having one component from each level in the hierarchy. In the household/person example a partition would consist of rows belonging to the same household. When a partition of data is presented to a user, redundancies should, of course, be suppressed.

One problem which may not be so difficult as it may seem is the problem of bringing corrections "back" from the target file to the files which are permanently stored in the data base. If the base operator approach is used consistently, it will always be possible to tell the **origin** of each item of data in the target file. In a sense this means that the sequence of base operators leading from the data base to the target file is **inverted.** For inversions to be possible, it is essential that metadata are properly maintained, as they should be in a base operator system.

Sometimes it may be practical not to carry out all editing on the final target file, but to perform each check on an original or intermediary file which is as normalized as possible with regard to the condition that is to be checked. Thus in our example, checks concerning individual persons only, could be made on the PERSON file, checks concerning households only, could be made on the HOUSEHOLD file, and only checks concerning combinations of households and persons would have to be made on the final target file, resulting from a join between PERSON and HOUSEHOLD.

The scheme may be summarized in a slightly more general way like this:

```
op INFILE_A <param expr> giving FILE_A1;

/some editing will possibly be done on A1, and the
resulting cor-rections will be brought back to infile
A/

op INFILE_B <param expr> giving FILE_B1;

/some editing will possibly be done on B1, and the
resulting corrections will be brought back to infile
B/

join FILE_A1, FILE_B1 <param expr> giving FILE_A2;
```

op FILE_A2 <param expr> giving TARGET_A3;

/remaining editing will take place on the target file A3, and the corrections will be brought back to infiles A and B/

Note. "op" represents an arbitrary combination of unary base operators to be performed on the file under consideration; "<param expr>" represents the parametrical information conveyed by the subclauses in the respective base operator statements.

Latest **R & D Reports (area ADB)** published by Statistics Sweden:

E-12 The OPREM-approach. An extension of an OPR-approach to include dynamics and classification (Erik Malmborg); 46 pp.
E-13 The Role of Databases in the Dessimination of Statistics (Lars Olsson); 9 pp.
E-14 An Analysis of Systems Design Methodologies using the ISO Framework (Erik Malmborg); 27 pp.
E-15 Using the RAPID Data Base Management System in Statistical Offices (Bo Sundgren); 8 pp.
E-16 Cell suppression in multi-dimensional frequency tables (Hans Block); 17 pp.
E-17 On Requirements for a Conceptual Schema Language (Björn Nilsson); 16 pp.
E-18 The Impact of Microcomputers on the Statistical Environment (Björn Nilsson); 18 pp.
-"- On the Usage of Microcomputers in Developing Countries (Björn Nilsson); 6 pp.
E-19 Conceptual Design of Data Bases and Information Systems (Bo Sundgren); 139 pp.
E-20 Stepwize formalization of information specifications by extending a simple object-oriented approach (Erik Malmborg); 20 pp.
E-21 Outline of an algebra of base operators for production of statistics (Bo Sundgren); 21 pp.
E-22 How to satisfy a statistical agency's need for general survey processing systems (Bo Sundgren); 11 pp.
E-23 A session with the CONDUCTOR (Bo Sundgren); 36 pp.
E-24 Useroriented Systems Development at STATISTICS SWEDEN (Bo Sundgren, Birgitta Lagerlöf and Erik Malmborg); 30 pp.
E-25 On the semantics of aggregated data (Erik Malmborg); 7 pp.
E-26 Using the base operator approach for editing of statistical data (Bo Sundgren); 11 pp.
E-27 Towards improving communications between statistical agencies - an evaluation of some communications options and details of STATCOM: A trial using 'com' computer conferencing and E-mail software (Jonathan Palmer); 11 pp.
E-28 The labour force survey in Zimbabwe - an illustration how the SCB model could be implemented in practice (Milan Sanovic); 53 pp.
E-29 An approach to the design of the time concept in the SCB model (Milan Sanovic); 24 pp.
E-30 The impact of the Development of EDP on Statistical Methodology and Survey Techniques (Lars Lyberg and Bo Sundgren); 26 pp.
E-31 SPORT-SORT - Sorting Algorithms and Sport Tournaments (Hans Block); 15 pp.

Copies of these reports may be ordered from **Statistics Sweden, att. Ingvar Andersson, S-115 81 Stockholm.**