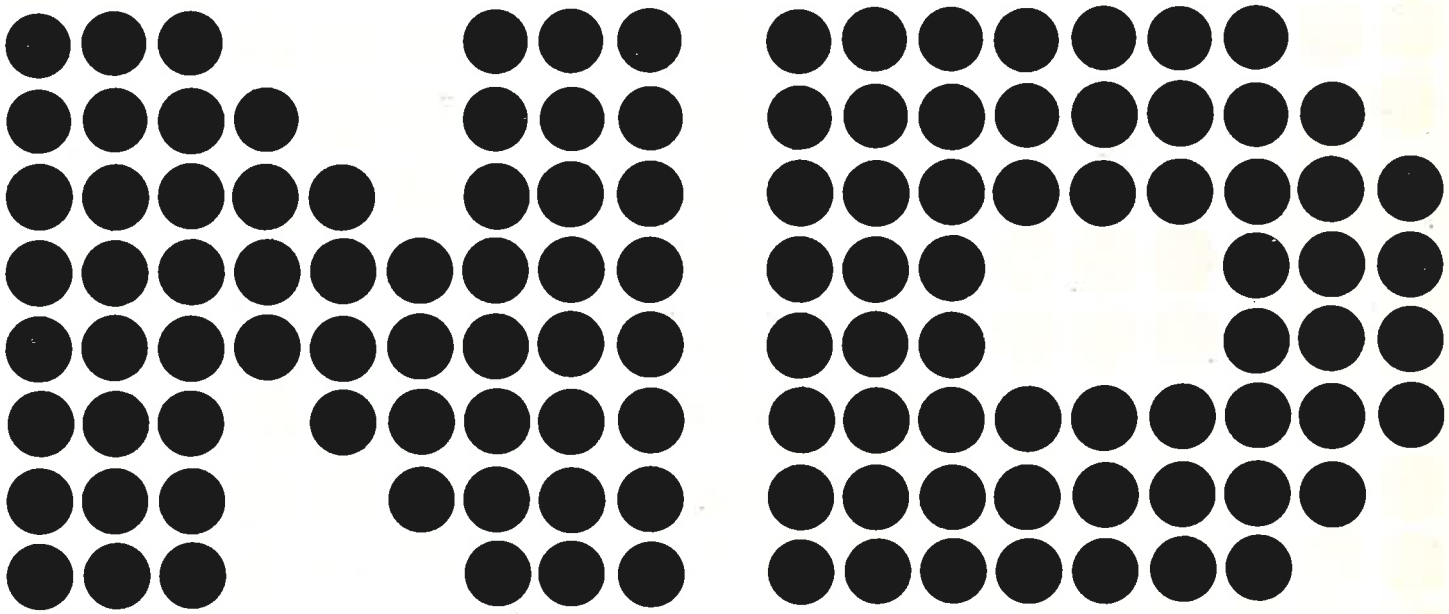
Specification of the Nord Query Language

NORSK DATA A.S



Specification of the Nord Query Language

23.1.1980

Ole Jorgen Hansen, Research & Development

This paper describes what Nord Query Language will look like from the user's point of view.

This paper is preliminary, and Norsk Data may change the language specifications without notice.

Contents

1. Summary

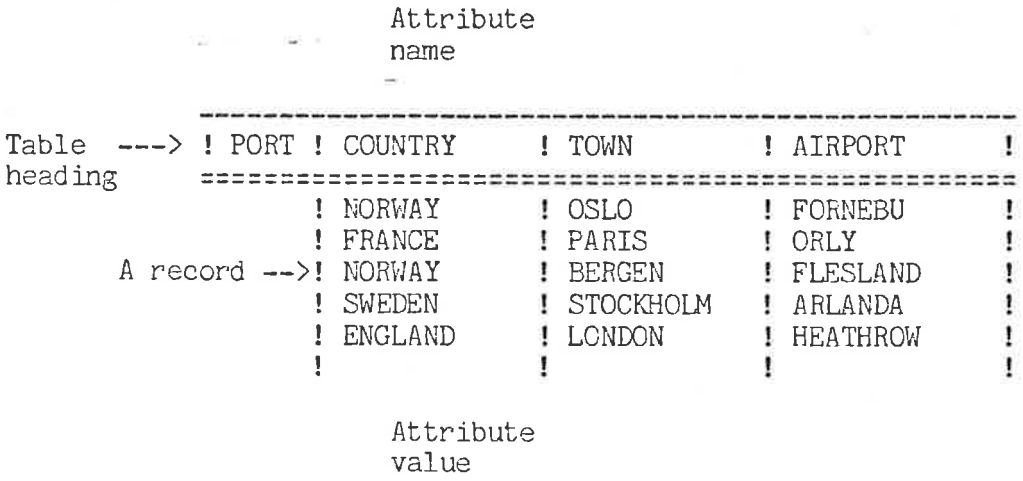
Query languages came into peoples minds after the relational data model was introduced in 1970. Before that people were working with report generators. The query language was therefore meant as a replacement for these. After 1970 the development has moved towards interactive computer systems. The query language was therefore given a new requirement. It should also be used interactively and had to be easy to learn.

Nord Query Language (NQL) is a high-level data base management language that provides a convenient and unified style to define, query and update a relational data base. The philosophy of Nord Query Language is to require the user to know very little in order to get started, and to make it easy to use the language without knowing the whole language syntax. The language syntax is simple, yet it covers both simple and complex queries.
Nord Query Language offers a report generator for the generation of reports. Once defined, queries and reports can be started and re-run.
Nord Query Language makes it possible to use a data base via a non-procedural language. The user doesn't need to know how the data is stored in the data base, he only needs to know what data the data base contains.
The user thinks of the data base as a set of tables. Each table has a set of attributes. He may operate on the data base by specifying table and attribute operations.
Experiments with languages of the same type have showed that they are easy to learn. It requires less than three hours of instruction for non-programmers to acquire enough skill to make fairly complex queries.

2. An introduction to the Relational Model.

In the relational model, a data base consists of a set of tables. Each table has a set of attributes.
A table can be understood as a file and the attributes as data items within the file. For those familiar with the SIBAS terminology, a table with attributes can be understood as a realm with items.

A table may look like this:



A row in the table is called a record, and the table may then be seen as a collection of records.
To be able to uniquely identify each record of a table, every table must possess a PRIMARY KEY. The primary key may consist of more than one attribute. For instance, in order to identify persons without using a person-number, both name and address together may be used. In the extreme, the primary key may contain all attributes of the table.
For all tables we have the following rules:

No two records are equal.

The ordering of the records within a table is immaterial.

All attributes are atomic. That is, they represent single values, such as numbers and character strings.

In the examples in the rest of this paper the following tables will be used. The primary keys are underlined.

CUSTOMER	(CUSTNO, NAME, ADDRESS, TELEPHONE)
ORDER	(CUSTNO, ORDERNO, DATE)
ORDERLINE	(CUSTNO, ORDERNO, ARTNAME, QTY)
ARTICLE	(ARTNAME, PRICE, STOCKQTY)
PARTS	(ARTSUP, ARTSUB, MADEQTY)
EMPLOYEE	(NAME, SALARY, MANAGER, DEPT)
DEPARTMENT	(DEPT, LOCATION)
FLIGHT	(FLIGHTNO, DEPARTURE, DESTINATION, DEPTTIME, ARRITIME)
AIRPORT	(CITY, AIRPORT)

3. An introduction to Nord Query Language.

A query language is a high level computer language which is oriented towards retrieval and maintenance of data from computer files or data bases.
Characteristically, the users of a query language are non programmers rather than programmers. The usage of a query language is on-line, but it should be possible to use it from batch too. A query language will be used for retrieval and update of data. The conditions for use are not predefined, but are created by the user in an ad-hoc manner. Nord Query Language can be viewed as a language for both information analysis and query handling. The language features are as follow:

Can be easily learned in a few hours.

Require little or no data processing background.

Allows the end user to express queries in familiar terminology and logic.

Minimizes the number of concepts to be learned.

Provides a powerful facility for answering complex queries.

Provides a report generator facility, which is a generalisation of the query facility.

Allows the user to define queries and reports that can be saved and executed at any later time.

The language covers a wide range of possibilities, and the highlights are:

Is nonprocedural and uses a two-dimensional visual format for formulating queries.

Enables the user to visualize the data base as user defined tables with attributes.

Provides a unified, consistent syntax with a few powerful operators.

Nord Query Language depends on and uses other program packages delivered by Norsk Data A.S. The specifications of the total system is:

Provides access to existing SIBAS data bases.

Provides access to SIBAS data bases defined and initiated from NQL.

Provides access to ASTRA data bases.
It will be possible to define data bases without using the ASTRAL language at all.

Provides for converting SIBAS data bases to ASTRAL data bases.

Provides queries and reports independent of the underlying

DBMS. This will at first be SIBAS, then ASTRA or SIBAS.

It will be catered for access via ISAM files. The query language will then be a single user version, because ISAM files do not allow concurrent access.

Provides for bulk loading of data bases from existing sequential files, and for load and unload of the data base.

Provides backup and recovery capabilities.

We have two kind of language concepts in Nord Query Language. These are queries and reports.

3.1. Queries.

A query is a sequence of commands defining an operation on the data base. In addition we can specify the format of input and output. This is optional, and the system will choose defaults.

```
<command-1>
<command-2>
<command-3>
   .....
   .....
<command-n>
<Specification of formats>
```

Some questions may be asked frequently. For such questions, it is possible to use variables in the query and give them a value at run time.
This will be described later.

3.2. Reports.

Reports can be defined with Nord Query Language too. We follow the method used for queries. In addition we have to describe the report layout. A report is written in the following way:

```
<command-1>
<command-2>
<command-3>
   .....
   .....
<command-n>
REPORTFORM
   <description of report layout>
ENDFORM
```

4. Input and output.

The Query Language will have an input and an output part. The first is used to specify a query, and the second is used for printing of the result.

The input must allow the user to specify a query in an understandable manner, and the output has to be easy to read. There are a number of alternative formats, but they all have essentially a table structure. We will use the tabular format, because it is easier to use this on a relational data base.

4.1. The tabular format.

The tabular format contains a heading consisting of table name and attribute names, and values for each record in rows. The following example shows a tabular skeleton with data.

```
-------------------------------------------------------------
! CUSTOMER ! CUSTNO ! NAME        ! ADDRESS      ! TELEPHONE !
=============================================================
          ! 101    ! JOE HOOLEY  ! LIVERPOOL    ! 120945    !
          ! 201    ! MIKE WOODS  ! LONDON       ! 320945    !
          ! 203    ! ADAM GREEN  ! MANCHESTER   ! 903193    !
          ! 504    ! CATY JONES  ! HULL         ! 574390    !
```

When the user defines a query he writes operations in empty tabular skeletons. The empty skeleton for the EMPLOYEE table looks like the following

```
-------------------------------------------------------------
! CUSTOMER ! CUSTNO ! NAME        ! ADDRESS      ! TELEPHONE !
=============================================================
          !        !              !              !           !
          !        !              !              !           !
```

The user fills in operations where he wants. If he wants to skip an attribute he writes <cr>. He can enter the navigate mode (as in TED) with <ctrl-S>, and can change the navigate mode with <cursor up> and <cursor down>.

When he is in the ATTRIBUTE navigation mode, <cursor left> and <cursor right> will move to the previous or next attribute in the skeleton. When he is in the CHAR navigation mode, the cursor will move within the field. When he is in the LINE navigation mode, <cursor left> and <cursor right> will move to the previous or next line in the skeleton.

We get out of the skeleton with the <home> character. The navigate mode is turned off with a <ctrl-S>.
When the user has filled in one line or moves one line down, the system will expand the skeleton with one empty line at the bottom. In this way it is possible to have many skeletons on the screen at the same time.

Later the user has the possibility to delete skeletons and lines, and to insert skeletons and lines in existing skeletons.

When the table table is larger than the width of the screen, some difficulties will occur.

There is one solution to this problem. We can split the table into many sub pictures on the screen. For instance, if we have a table containing 6 attributes with 20 characters each, we can write the table as three pictures. This operation is called horizontal scrolling.

Example:

The table

```
-------------------------------------------------------------------------------
! TEST ! ATTR1    ! ATTR2    ! ATTR3    ! ATTR4    ! ATTR5    ! ATTR6    !
===============================================================================
       ! .        ! .        ! .        ! .        ! .        ! .        !
       ! .        ! .        ! .        ! .        ! .        ! .        !
       ! .        ! .        ! .        ! .        ! .        ! .        !
       ! .        ! .        ! .        ! .        ! .        ! .        !
```

will be printed as

```
-----------------------------------------
! TEST ! ATTR1    ! ATTR2    ! ATTR3    !
=========================================
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
```

We scroll the picture to the left and get

```
-----------------------------------------
! TEST ! ATTR3    ! ATTR4    ! ATTR5    !
=========================================
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
```

We make a new scroll to the left and get

```
-----------------------------------------
! TEST ! ATTR4    ! ATTR5    ! ATTR6    !
=========================================
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
       ! .        ! .        ! .        !
```

The user can control the horizontal scrolling with the function keys <cursor left> and <cursor right>.

5. The Query Language.

All programming is done within two-dimensional skeleton forms. This is accomplished by filling in the appropriate tables with an example of the solution.

5.1. Basic consepts.

We have four different types of variables in a query. The distintion is made between a constant element, an example element, a table operator and a command element.

5.1.1. Example elements.

Example elements are names we give any variable from the data base. They are identifiers prefixed with the underscore character.

Example:

```
_EXAMPLE
_SALES
```

They are used for linking the subquestions in a query together and for expressions in constant elements.

An example element can be used when we want to perform complex queries on the data base. It is possible to assign values to an example element and use it as a selection criteria.

Example:

```
_NAME   = 'JEAN WEBSTER' OR 'JIM SMITH'
_SALARY =  >80000 AND <100000
```

In this case we use the example element as a condition box.

We have another use of a condition box too. Often we want to calculate something using the result from many attributes. This can be done using example elements. In addition it is possible to use the functions MINIMUM, MAXIMUM, AVERAGE, SUM and STDEV.
If we write the following condition box

```
_TOTAL = SUM(_QTY)
```

the example element _TOTAL will be updated according to the expression each time _QTY gets a value from a new record.
It is also possible to use expressions within the functions.

Example:

```
_SUM    = SUM(_NO * _PRICE)
_MINCOST = MINIMUM(_COST)
```

5.1.2. Constant elements.

Constant elements are numbers and character strings. They are used for selection of records from the tables.

Example:

```
125000
'JEAN PAUL GEOFFRY'
'PAUL JONES'
```

If we have a string containing letters only we can write it without the string quotes. The string must not be equal with the names of the functions described below.
We may then write

```
'JONES' or JONES
'RESEARCH' or RESEARCH
```

As selection criteria one may use example elements and constant elements or a combination of these.

We may prefix these with the following operators:

=	equal
><	not equal
>	greater than
>=	greater or equal
<	less than
<=	less or equal

If nothing is mentioned there is an implied equality.

We may also use the following functions to specify complex operations. They operate on the attribute where they are mentioned.

MINIMUM.	Finds smallest attribute value
MAXIMUM.	Finds largest attribute value
SUM.	Computes the sum of all the attributes
COUNT.	Counts the number of occurences
AVERAGE.	Computes the average of all the attributes
STDEV.	Computes the standard deviation of all the attributes
BETWEEN.	Finds all records with attribute values in a certain range

If we want the minimum salary for the EMPLOYEE table, we can specify it in the following way. We use the constant element MINIMUM in order to find the record with the minimum salary.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
PRINT.		MINIMUM.		

If we wants the minimum salary for the SUPPORT department, we have to specify an additional selection.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
PRINT.		MINIMUM.		SUPPORT

In addition it is possible to use the arithmetic operators +, -, * and /. These can be used to calculate the wanted selection expression. We may also use paranthesis.

Example:

```
_SALARY+10000
_OLDSALARY*1.15
(_OLDSALARY+_LOMAX)*1.07+5000
```

It is possible to use variables for selection of records. These variables are identifiers prefixed with a question mark, and can be given a value at run time. Thus it is possible to make very flexible queries. They are called question elements.

Example:

```
?DEPT
?SALARY
```

In this way it is possible to define queries and give values to the question elements at run time.

5.1.2.1. String manipulation.

Often the user is interested in finding attributes starting, ending or containing a special letter combination. This can be described in an easy manner with a constant element.

When we have a string enclosed by underscores, we will search for a textual equlity. When we use a question mark we skip a letter.

Example:

_HA_	will find all strings starting with HA
_HA_XXX	will do the same. The 3 X's tell us that someting may follow. These may be omitted
_HA_XX_SEN_	will find all string starting with HA and ending with SEN. F.ex HANSEN and HANSSEN. The 2 X's tell us that something may come in between.
_H_?_N_	will find all strings with first character H and third character N. F.ex HANSEN and HENSON

5.1.3. Table operators.

A table operator tells which operation we want to do on a record. We have the following operators

INPUT.	We want to insert a new record
UPDATE.	We want to update records
DELETE.	We want to delete records
PRINT.	We want to print all attribute values in a record

The commands can be abbriviated. Thus it is possible to write I., P. and so on.

Sometimes the user wants to reduce the output from a given query. It is possible to specify the number of records wanted in the output. This is done specifying the wanted number of records in the table operator. The number of records wanted must occur after the PRINT. operator.

Example:

```
PRINT.10
P.6
```

Example:

Find any 8 employees with manager JONES.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
PRINT.8			JONES	

We can also use a question element, and give the number of records we want to retrieve at run time.

5.1.4. Command elements.

A command element tells what to do with an attribute. The following commands are used

PRINT.	Print the attribute value
UNIQUE.	Find all unique records, i.e the attribute value can not occur more than once
ASCENDING.	Sort the records on ascending attribute value
DESCENDING.	Sort the records on descending attribute value

5.1.4.1. Sorting of records.

In the relational model the records in a tuple is not sorted in any order. If the user wants this to be done, he has to specify it in a query. It is possible to sort the result in ascending or descending order. As it is possible to sort on many criteria at the same time, the priority of the sort key has to be given.
The priority is a number enclosed by parenthesis. If the priority is omitted, (1) is assumed.

If we write ASCENDING(1), it is the first key. ASCENDING(2) is the second key and so on.

5.1.4.2. Selection of unique records.

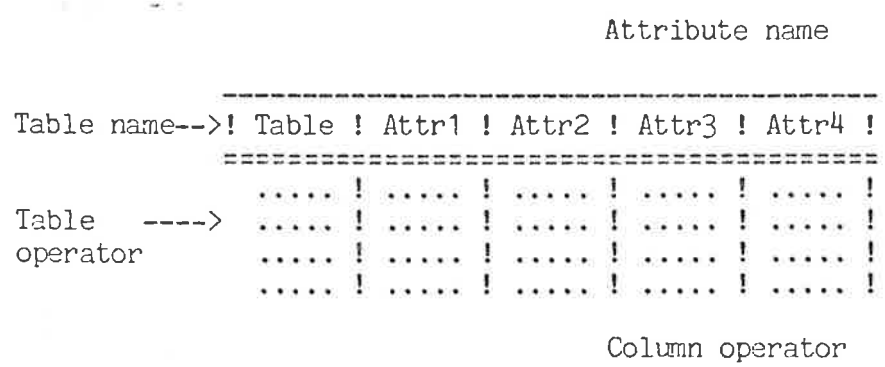
When we operate on parts of a table we will often get duplicates. Because it is very expensive to remove duplicates, the user has the option of specifying whether a unique result is wanted or not. This is done with the UNIQUE. operator.

With these basic concepts, the user can express a wide variety of queries. To perform operations on the data base, the user fills in an example of a solution to that operation in blank skeleton forms. These are associated with the actual tables.

The column operator is a combination of a command element, constant element and an example element. They must follow in the given sequence, and all of them may be omitted.

A query is given by filling in table and column operators.

The skeleton will look like the following:



5.2. Examples of use.

We show some examples to clarify how queries are written. The empty skeleton table will be written by the system on request, and the user fills in the wanted operation. How the user specifies which table skeleton he wants to be written on the screen, is described later.

Find name and salary for all employees working in the SALES department.

We achieve this selecting all records in the EMPLOYEE table with department equal SALES.

When all records are found, we print the name and salary attributes

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
	PRINT.	PRINT.		SALES

The result is:

EMPLOYEE	NAME	SALARY
	JONES	96.000
	MILLS	84.600
	JOHNSEN	102.000

Find name and salary for all employees working in departments located in STRASBOURG.

First we have to find all departments located in STRASBOURG. These are stored in the example element _SALES.
Then we select all employee records with department name equal to the values assigned to the example element _SALES. When all records are found, the name and salary will be printed.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
	PRINT.	PRINT.		_SALES

DEPARTMENT	DEPT	LOCATION
	_SALES	STRASBOURG

The result is:

EMPLOYEE	NAME	SALARY
	JOHNSEN	100.000
	HANSEN	90.000
	BEAM	180.000
	WILEY	85.000
	JONES	96.000
	OLSEN	54.000

Sometimes we have to use computed values to find a result. This can be done using expressions, and example elements can be used as ordinary variables.

Find the name of all employees earning more than 10000 more than WILEY.

First we find WILEY's salary and assign this to the example element _SAL.
Then we find all employee records with salary greater than the value of the example element _SAL plus 10000.
In this case we se how an example element can be used in an expression.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
	PRINT.	>(_SAL+10000)		
	WILEY	_SAL		

The result is:

```
-------------------------
! EMPLOYEE ! NAME       !
=========================
           ! JOHNSEN    !
           ! BEAM       !
           ! JONES      !
           ! WILSON     !
```

Sometimes it is desireable to write a long string of operations for a column operator. We may then write the commands on several lines. The string starts and ends with the underscore character ^.

Find all departments sorted ascending and remove all duplicates.

In this case we want to select all unique department names. If we had used the PRINT operator alone, we had got all values in the DEPT column of the table. Thus we have to specify the UNIQUE operator in order to remove all duplicates. The ASCENDING operator will sort the selected department names.

```
-----------------------------------------
! DEPARTMENT ! DEPT         ! LOCATION   !
=========================================
            !^PRINT.        !            !
            ! UNIQUE.      !            !
            ! ASCENDING. ^!            !
            !              !            !
```

The result is:

```
----------------------------
! DEPARTMENT ! DEPT        !
============================
            ! DEVELOPMENT !
            ! ECONOMY     !
            ! SALES       !
            ! SUPPORT     !
```

It is possible to link many constant elements together in a condition box. Here we may describe complex selection expressions by the use of AND and OR.

Find the name of all employees working in the SALES department having GIBBS or JONES as manager.

We assign the values GIBBS and JONES to the example element _MAN, and use it in a selection expression.

In order to get the skeleton for the condition box printed on the screen, we use the <new skeleton> command with the system table CONDITION as parameter.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
           ! PRINT.    !           ! _MAN       ! SALES     !
           !           !           !            !           !

-----------------------------------------------
! CONDITION !                                 !
===============================================
            !_MAN  = 'GIBBS' OR 'JONES'       !
            !                                 !
```

We show an example to clarify the use of question elements. We want a query finding all employees with salary in a certain range. We use the BETWEEN operator, and use question elements as parameters to this.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
  PRINT.   !           !^BETWEEN   !            !           !
           !           !(?LOWSAL,  !            !           !
           !           ! ?UPPSAL).^!            !           !
           !           !           !            !           !
```

When the query is executed the system will ask for a value to LOWSAL and UPPSAL. The user will at the same time see how the variables are used.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
  PRINT.   !           !BETWEEN    !            !           !
           !           !(.....     !            !           !
           !           !.....)     !            !           !
           !           !           !            !           !
```

The user enters these and the query will continue.

We want a query finding the cost of the orders belonging to JONES in LONDON.

This is a very complex query, and needs some explaination. First of all we have to find the wanted customer. This is done with a selection on name and address. The customer number of the found customer is stored in the example element _NO.
_NO is then used for selecting all ORDERLINE records belonging to the customer. For each found record we find the ordered article and the number of articles ordered. These values are stored in the example elements _ART and _ANT.
Then we find the price of the article _ART and store it in the example element _PR.
We have now all information we need about the orderline and the cost is calculated. The total is summed up in the example element _TOTAL.

```
-------------------------------------------------------------------------
! CUSTOMER ! CUSTNO ! NAME            ! ADDRESS         ! TELEPHONE !
=========================================================================
           ! _NO    ! JONES           ! LONDON          !           !
           !        !                 !                 !           !

-------------------------------------------------------------------
! ORDERLINE ! CUSTNO ! ORDERNO ! ARTNAME            ! QTY    !
===================================================================
             ! _NO    !         ! _ART               ! _ANT   !
             !        !         !                    !        !

-----------------------------------------------------
! ARTICLE ! ARTNAME              ! PRICE   ! STOCKQTY !
=====================================================
          ! _ART                 ! _PR     !          !
          !                     !         !          !

-------------------------------------------------
! CONDITION !                                   !
=================================================
  PRINT.    !_TOTAL = SUM( _ANT * _PR )         !
            !                                   !
```

The query may then give the following result

```
--------------------------------
! CONDITION !                  !
================================
            ! 13.654,45        !
```

We want a query listing all employees sorted on ascending department, manager and salary.

We use the sort operator in order to achieve this.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
PRINT.		ASCENDING(3).	ASCENDING(2).	ASCENDING(1).

5.3. Update of records in the data base.

Many of the proposed query languages cater only with retrieval of data from the data base.

We want the user of the query language to be able to do everything it is possible to do with application programs.

Thus it must be possible to insert, update and delete records in the data base.

5.3.1. Insertion of records.

When we want to insert a record into the data base, we have to specify INSERT. as table operation. We have to give values to all attributes. The values may be taken from records in the same table or from records in other tables.

Insert a new employee JONES in the RESEARCH department. His salary is 120000 and the manager is CARTER.

We use table operator INSERT., and specify the attribute values in constant elements.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
INSERT.	JONES	120000	CARTER	RESEARCH

Insert a new employee OLSEN and give him the same salary, manager and department as WILEY. The same goes for the new employee WILSON, but his salary shall be equal to the average salary.

First we find the salary, manager and department of WILEY and assign these values to the example elements _SAL, _MGR and _DEP.
These example elements are then used as constant elements in the isertion.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
INSERT.	OLSEN	_SAL	_MGR	_DEP
	WILEY	_SAL	_MGR	_DEP
INSERT.	WILSON	AVERAGE.	_MGR	_DEP

5.3.2. Update of records.

When we want to update records, we have to specify UPDATE as table operator. We have to find the records to be updated before the update can take place. This is done by an example element.

Give all employees in the SALES department a 10% pay rise.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
  UPDATE.  ! _WHO      ! 1.1*_OLD  !            !           !
         - ! _WHO      ! _OLD      !            ! SALES     !
           !           !           !            !           !
```

This way of writing the query is all right. It is, however, possible to write that query in an easier way. When we are going to update an attribute relative to its old value, we may drop the example element. We then have the following way of writing:

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
  UPDATE.  !           !  *1.1     !            ! SALES     !
           !           !           !            !           !
```

All employees with manager SMITH will get a new manager WEBBS.

We find all employees with manager SMITH and store them in the example element _PERSON. Then we give these employees a new manager.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
  UPDATE.  ! _PERSON   !           ! WEBBS      !           !
           ! _PERSON   !           ! SMITH      !           !
           !           !           !            !           !
```

5.3.3. Deletion of records.

When we want to delete records, we have to specify DELETE as table operator. We have to find the records to be deleted before the delete can take place. This can be done with constant elements or example elements.

Delete all employees working in the REPAIR department and all working in departments located in LONDON.

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
DELETE.				REPAIR
DELETE.				_WHERE

DEPARTMENT	DEPT	LOCATION
	_WHERE	LONDON

Delete all orders ordered before 110979 belonging to customers in PARIS.

CUSTOMER	CUSTNO	NAME	ADDRESS	PHONE
	_NO		PARIS	

ORDER	CUSTNO	ORDERNO	DATE
D.	_NO		<110979

In this case the ORDERLINE records belonging to the deleted ORDER records will be deleted.

6. The different user interfaces.

Up to now we have only mentioned the user interface using screen pictures. We have a linear interface to the Query Language too. This permits input of queries in a one-dimensional string format. The way of expressing a query is the same, the difference is the way of writing it. The user has to write table and attributenames.

Instead of filling in a skeleton form, the user writes a string containing the same information.

We have to write a colon behind the table name if there is a table operator, and a colon behind each attribute name.

Example:

The query

EMPLOYEE	NAME	SALARY	MANAGER	DEPT
	PRINT.	>_S1	_BOSS	
	_BOSS	_S1		

is equivalent to the linear form

```
EMPLOYEE(NAME:PRINT.,SALARY:>_S1,MANAGER:_BOSS)
EMPLOYEE(NAME:_BOSS,SALARY:_S1)
```

Each interface has some keywords for defining the start and end of a query, and some facilities for editing, storing, deleting and execution of the query.
Note that reports are created using the linear interface. The same applies for the definition of input and output forms.

When we write a condition box in the linear interface, we have to end it with a semicolon.

Example:

```
_TOTAL = SUM( _SUM );
_PRICE = _ANT * _COST;
```

6.1. Picture oriented interface.

A query is written as pictures on the screen. The user is able to define, edit, execute, store and delete queries. We enter the different modes with the following commands.

```
!                                                                          !
! Nord Query Language help commands                                        !
!                                                                          !
!     Command    Function                                                  !
!                                                                          !
!     H or ?   -      Help mode                                            !
!     D                Query definition                                    !
!     E                Execute query                                       !
!     S                Store query                                         !
!     R                Delete query                                        !
!     I                Insert in table                                     !
!     M                Modify query                                        !
!     F                First page                                          !
!     <cursor-down>    Next page                                           !
!     <cursor-up>      Previous page                                       !
!     L                Last page                                           !
!     X                eXit from NQL                                       !
!     <home>           Enter command mode                                  !
!                                                                          !
!                                                                          !
!                                                                          !
!--------------------------------------------------------------------------!
! Command mode                                                             !
!                                                                          !
----------------------------------------------------------------------------
```

In the examples in the paper the below decscribed commands are given. They correspond to the commands mentioned above. They are only written in words. For some of the commands the system will as for parameters. We write these after the commands in the examples. For instance

```
<new skeleton> EMPLOYEE
```

will print a skeleton of the EMPLOYEE table on the screen.

We have the following commands:

<define query>
The definition of a query starts here.

<exit query>
The definition of a query ends here. Everything from the last <define query> is included.

<new skeleton>
We want to get a skeleton table of a certain table. The system will ask for the name. The skeleton will be used until a new skeleton is wanted.

<new virtual table>
We want to get an empty skeleton on the screen. We can then define the wanted table and attribute names and can operate on the virtual table.

<execute query>
We want to execute a query or a report. The system will ask for the name. If we want the last query, we enter a blank name. The system will ask for input file and output file, and Default is TERMINAL.

<modify query>
We want to edit a query or a report. The system will ask for the name. If we want the last defined query we, enter a blank name. The edit modus will use a system a la TED/PED. It will not be described further here.

<store query>
We want to store the last defined query or report. The system will ask for the name to be given to this query.

<delete query>
We want to delete a stored query or report. The system will ask for the query name.

<help mode>
The user wants to retrieve information about the data base and the legal commands.

Example:

We want to define a query, LIST_EMPLOYEES, finding all employees working in a certain department.

```
<define query>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
   PRINT.  !           !           !            ! ?DEPART   !
           !           !           !            !           !

<exit query>
<store query> LIST_EMPLOYEES
<run query>
Name:
Input-file:  TERMINAL
Output-File: TERMINAL

-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY    ! MANAGER    ! DEPT      !
=============================================================
   PRINT.  !           !           !            !.........!
           !           !           !            !           !
```

6.2. Linear interface.

A query or report is written as a string containing all information about the data base operations. It is possible to define, edit, store, delete and run queries and reports using commands. We enter the different modes using the same commands mentioned for the screen oriented interface. We can not use the <new skeleton> command in this case.
In order to control the definition of queries, reports and input and output forms, we use some commands. These are written in the string defining the query.

BEGIN
The definition of a query starts.

END
The definition of a query ends here. Everything from the last BEGIN is included.

VIRTUALTABLE
We can define virtual tables.

ENDVIRTUAL
We end the definition of virtual tables.

INPUTFORM
Starts an input form definition.

OUTPUTFORM
Starts an output form definition.

REPORTFORM
Starts the definition of a report format.

ENDFORM
Ends a form definition.

Example:

We want to define the query mentioned above in the linear format.

```
<define query>

BEGIN
   EMPLOYEE:PRINT.(DEPT:?DEPART)
END

<exit query>
<store query> LIST_EMPLOYEES
<run query>
Name:
Input-file:  TERMINAL
Output-file: TERMINAL
DEPART: ....................
```

7. Definition of forms.

For queries often used it is suitable to have specially designed input and output formats. This can be done by defining forms. The definition of forms will be done in the same way as in Nord Screen Handling System today. Instead of the definition of the fields, the user enters question elements used in the query. The system will then know the type of each field.

Example:

We want to define a query inserting ORDER-tuples into the data base. We want the input form to be specially designed.

```
<define query>
BEGIN
   ORDER:INPUT.(CUSTNO:_CNO,ORDERNO:?ONO,DATE:?DATE)
   CUSTOMER(CUSTNO:_CNO,NAME:?CNAME,ADDRESS:?CADDR)
   INPUTFORM

      Transaction for insertion of
      order records
      ----------------------------

      Today's date: ?DATE

         Customer's name:    ?CNAME
         Customer's address: ?CADDR

              Order number:  ?ONO

   ENDFORM
END
<exit query>
<store query> NEW_ORDER
```

Everything written in the form definition will be placed absolutely on the screen, i.e. as it is written.
When the query is executed the following picture will be written on the screen. The user enters the values, and the query will be executed.

```
      Transaction for insertion of
      order records
      ----------------------------

      Today's date:  ......

         Customer's name:    ..............................
         Customer's address: ..................

              Order number:  ......
```

8. The Data Base Administrator module.

Nord Query language will be used by all users including the Data Base Administrator (DBA). DBA's responsibility is to create and maintain data bases. Nord Query Language has a module specially designed for these purposes. It has the following features:

Definition of a relational data base.
All information about the tables will be described. The legal use of the tables are defined as integrity constraints. The legal values of the attributes are defined too.

Definition of user submodels.
Each user can use a subset of the tables in the data base and may have his own legal usage of them. It is also possible to create virtual tables. These are tables with information from many tables. The virtual tables can only be used for retrieval of information.

Definition of SIBAS data base.
All information about the underlying SIBAS data base is given. This information is the realms, items, index and calc keys and sets used in the data base.
If this information is omitted the system will generate a schema for the SIBAS data base from the relational data base.

The DBA module will be used as the ordinary NQL language. The user enters the different modes, and describe the data base by filling in empty skeleton tables. We enter the different modes by some commands. These are:

<define model>	- the relational data model is defined
<define submodel>	- a relational submodel is defined
<end of (sub)model>	- end of the model or submodel definition
<list model>	- list the model description
<list submodel>	- list the submodel description
<export tables>	- export tables from a model or submodel to another submodel
<import tables>	- import tables from the above model or submodel
<define types>	- define an attribute type
<list types>	- list the defined attribute types
<define table>	- define a new table
<list tables>	- list the existing tables
<define table link>	- define a table link for a virtual table
<list table link>	- list a table link
<new skeleton>	- print a table skeleton to the screen
<define constraint>	- define a new constraint for operations on the data base
<list constraint>	- list the defined constraints for a specific or all tables

<access (sub)model>	- define which model or submodel the users are allowed to access
<define data base>	- describe a SIBAS data base to the system
<end data base>	- end the data base description
<list data base>	- list the data base description for a specific data base
<define realm>	- describe the realms used
<list realm>	- list all the realms
<define calc>	- describe the calc keys
<list calc>	- list the calc keys
<define index>	- describe all indexes used
<list index>	- list all indexes used in a data base
<define set>	- describe all sets used
<list set>	- list all sets
<define SIBAS link>	- describe the mapping between the relational data base and the SIBAS data base
<list SIBAS link>	- list the mapping description for a virtual table

8.1. The model definition.

NQL deals with the following objects:

TABLE	- a relational table
ATTRIBUTE	- an attribute of the table
TYPE	- the type of an attribute
DOMAIN	- the value range of a type

A data base may be defined by ASTRAL as well as NQL. Because of that we have to ensure that an ASTRAL type can be translated into a NQL type, and vice versa.

8.1.1. Attribute name.

The attribute name is a string of up to 16 character including the underscore character. We have chosen 16 characters because ASTRAL at current time uses only 12 characters.

Example:

```
DATE_ORDERED
CUSTOMER_NO
ORDERNO
NAME
```

8.1.2. Attribute types.

The attribute type is a string describing the format of the attribute and the legal attribute domain. The attribute type is used for syntax checks of the queries. When we have attributes connected together via example elements, the attribute types must be the same. Thus it possible and meaningfull to connect an order with its order detail via an order number. It is not meaningfull to connect a number with salary as these are attributes with different types.
The description of the attribute format follows a syntax simular to COBOL. In addition we may use INTEGER and REAL.

Example:

9(3)	- An integer with 3 digits.
A(15)	- An alphanumeric string of length 15.
X(15)	- A character string of length 15.
INTEGER	- A standard integer.
REAL	- A standard real.
999,999	- An integer with 6 digits. The three last digits are seperated with a comma. F.ex. 123,000

The DBA may specify legal values for the attributes. For strings it is possible to specify the legal characters. For integers and reals it is possible to specify a legal range.

Example:

```
0 : 230000
'A' : 'Z' , '-'
123.00 : 134.00
```

8.1.3. Definition of integrity constraints.

If a user uses a data base incorrectly, he may get peculiar results. The purpose of integrity constraints is to describe how the data base should be used, so that misuse is detectable. The integrity constraints will be specified for insertion, deletion and update. At the same time we may give the error message we want to be written when an integrity constraint is broken.

8.1.3.1. Insertions.

Before a record can be inserted all key attributes must have a value assigned, and the values must satisfy the legal attribute value. It is possible to give non-key attributes a null-value. For strings the null value is blanks, for integers and reals the null value is zero. These are defaults, and the user can define them for each type. Often there is a logical connection between tables. A record in one table has to exist before a record in another table can be inserted. This constraint is given by specifying that the common attribute in the two tables has to exist.
For instance, an EMPLOYEE record can not be inserted if the DEPARTMENT record with the same DEPT as the EMPLOYEE record doesn't exist. The new employee is not allowed to earn more than his manager. We specify this demand with the following NQL-statement.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY     ! MANAGER    ! DEPT      !
=============================================================
    I.    !           ! <_SAL      ! _BOSS      ! _DEPT     !
          ! _BOSS     ! _SAL       !            !           !

-----------------------------------------
! DEPARTMENT ! DEPT      ! LOCATION      !
=========================================
            ! _DEPT     !               !
            !           !               !
```

8.1.3.2. Deletion.

When we delete a record from the data base, we may get some undesireable effects If we have two tables that are logically connected, the deletion of a record in one of them may result in an inconsistent data base.
We must therefore have the possibility to specify the demands which have to be satisfied before a deletion can be made. For instance, a DEPARTMENT record can not be deleted if there are EMPLOYEE records with the same DEPT. We specify this with the following NQL statement.

```
-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY     ! MANAGER    ! DEPT      !
=============================================================
          !           !            !            ! ><_DEPT   !
          !           !            !            !           !

-----------------------------------------
! DEPARTMENT ! DEPT      ! LOCATION      !
=========================================
       D.   ! _DEPT     !               !
            !           !               !
```

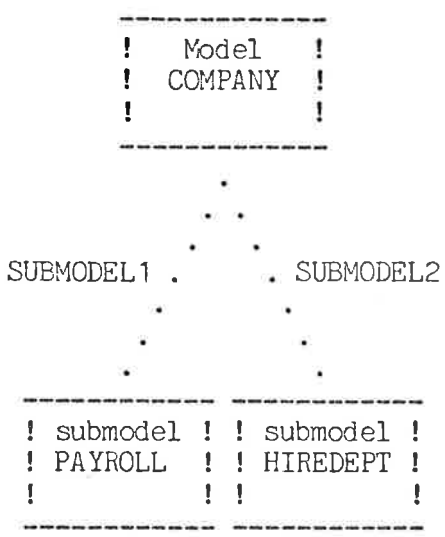
8.1.3.3. Update.

Before we can update a record the same demands that go for deletion and insertion have to be satisfied. In addition we may specify other demands for the new record. For instance, we may specify that the new salary has to be greater than the old salary. If we want the new salary to be at least 15% greater than the old, we may write the following NQL statement.

```
-------------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY       ! MANAGER    ! DEPT      !
===================================================================
    U.    ! _PERSON   ! >1.15*_SAL   !            !           !
          ! _PERSON   ! _SAL         !            !           !
```

8.2. The submodel definition.

Usually a user is allowed to use some parts of the data base. The DBA must therefore have the possibility to give each user a submodel. The users may also have different usage modes. For instance, not everybody are allowed to go and update the information in the EMPLOYEE table.
Sometimes a user wants to get information from many tables. This can be done in an easy manner by defining virtual tables in a sub-model. The system will then retrieve the information for the user. Note that a virtual table never can be used for other tasks than retrieval. This is because a view may be taken from more than one table, and update, deletion and insertion will be impossible to execute.
The definition of submodel can be done in many steps. It is possible to have submodels of a submodel and so on.
We show the following example with a model and two submodels.



A model with submodels.

8.3. Example of use.

We want to define a data base containing the EMPLOYEE and DEPARTMENT tables. We give some legal attribute values, and specify demands for insertion, deletion and update.

```
<define model> COMPANY
<define type>
```

TYPELIST	TYPE	FORMAT	DOMAIN
	NAMES	A(20)	'A':'Z','-'
	DEPARTMENT	A(15)	'A':'Z'
	SALARIES	9(6)	50000:230000
	TOWNS	A(25)	'A':'Z','-'

```
<define table> EMPLOYEE
```

ATTRIBUTES	NAME	TYPE	KEY
	NAME	NAMES	YES
	SALARY	SALARIES	NO
	MANAGER	NAMES	NO
	DEPT	DEPARTMENT	NO

```
<define table> DEPARTMENT
```

ATTRIBUTES	NAME	TYPE	KEY
	DEPT	DEPARTMENT	YES
	LOCATION	TOWNS	NO

```
<export tables> SUBMODEL1

-------------------------------------------------
! EXPORTLIST ! TABLE          ! OPERATION        !
=================================================
             ! EMPLOYEE       ! P.               !
             !                !                  !

<export table> SUBMODEL2

-------------------------------------------------
! EXPORTLIST ! TABLE          ! OPERATION        !
=================================================
             ! EMPLOYEE       ! P.,I.,U.,D.      !
             ! DEPARTMENT     ! P.,I.,U.,D.      !
             !                !                  !

<define constraint>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME     ! SALARY     ! MANAGER    ! DEPT     !
=============================================================
     I.    !          !            !            ! _DEPT    !
           !          !            !            !          !

<new skeleton> DEPARTMENT

-----------------------------------------
! DEPARTMENT ! DEPT      ! LOCATION     !
=========================================
             ! _DEPT     !              !
             !          !              !

<end constraint>
Give errormessage: >>> The department does not exist <<<
<define constraint>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME     ! SALARY     ! MANAGER    ! DEPT     !
=============================================================
     I.    !          ! <_SAL      ! _MAN       !          !
           ! _MAN     ! _SAL       !            !          !

<end constraint>
Give errormessage: >>> It is illegal to earn more than your manager <<<
```

```
<define constraint>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY     ! MANAGER    ! DEPT      !
=============================================================
     I.    ! ><_EMP    !            ! _EMP       !           !
           !           !            !            !           !

<end constraint>
Give errormessage: >>> You can not be your own manager <<<
<define constraint>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME _     ! SALARY     ! MANAGER    ! DEPT      !
=============================================================
           !           !            !            ! _DEPT     !
           !           !            !            !           !

<new skeleton> DEPARTMENT

-----------------------------------------
! DEPARTMENT ! DEPT      ! LOCATION      !
=========================================
        D.   ! _DEPT     !               !
             !           !               !

<end constraint>
Give errormessage: >>> There are employees working in the department <<<
<define constraint>
<new skeleton> EMPLOYEE

-------------------------------------------------------------
! EMPLOYEE ! NAME      ! SALARY       ! MANAGER    ! DEPT      !
=============================================================
     U.    ! _PERSON   ! >1.15*_SAL   !            !           !
           ! _PERSON   ! _SAL         !            !           !

<end constraint
Give errormessage: >>> New salary must be 15% larger than the old one <<<
```

<define data base>
<define realm>

REALM	NAME	ITEM	LENGTH
	DEPARTM	DEPT	15
	DEPARTM	LOCATION	25
	EMPLOYEE	NAME	20
	EMPLOYEE	SALARY	4
	EMPLOYEE	MANAGER	20
	EMPLOYEE	DEPT	15

<define index>

INDEX	REALM	ITEM	AUTOMATIC
	EMPLOYEE	NAME	YES
	EMPLOYEE	MANAGER	YES
	DEPART	DEPT	YES

<define set>

SET	NAME	OWNERREALM	OWNERITEM	MEMBERREALM	MEMBERITEM
	DEPTEMPL	DEPART	DEPT	EMPLOYEE	DEPT

<define SIBAS link>

LINK	TABLE	ATTRIBUTE	REALM	ITEM
	EMPLOYEE	NAME	EMPLOYEE	NAME
	EMPLOYEE	SALARY	EMPLOYEE	SALARY
	EMPLOYEE	MANAGER	EMPLOYEE	MANAGER
	EMPLOYEE	DEPT	EMPLOYEE	DEPT
	DEPARTMENT	DEPT	DEPART	DEPT
	DEPARTMENT	LOCATION	DEPART	LOCATION

<end data base>
<end of (sub)model>

We define two submodels PAYROLL and HIREDEPT. Users using the PAYROLL submodel are allowed only to retrieve information from the data base. Users using the HIREDEPT have no restrictions of usage.

```
<define submodel> PAYROLL

Refers to (sub)model: COMPANY
Model-password       : SUBMODEL1

<import tables>

-------------------------------------------------
! IMPORTTABLE ! TABLE          ! OPERATION      !
=================================================
             ! EMPLOYEE       ! P.             !
             !                !                !

<end of (sub)model>
<define submodel> HIREDEPT

Refers to (sub)model: COMPANY
Model-password       : SUBMODEL2

<import tables>

-------------------------------------------------
! IMPORTTABLE ! TABLE          ! OPERATION      !
=================================================
             ! EMPLOYEE       ! P.,I.,U.,D.    !
             ! DEPARTMENT     ! P.,I.,U.,D.    !
             !                !                !

<define table> EXPANDEMP

-----------------------------------------------------------
! ATTRIBUTES ! NAME           ! TYPE           ! KEY !
===========================================================
             ! NAME           ! NAMES          ! YES !
             ! SALARY         ! SALARIES       ! NO  !
             ! DEPT           ! DEPARTMENT     ! YES !
             ! LOCATION       ! TOWNS          ! NO  !
             !                !                !     !
```

```
<define table link>
<new skeleton> EXPANDEMP

-------------------------------------------------------------------------
! EXPANDEMP ! NAME          ! SALARY      ! DEPT        ! LOCATION    !
=========================================================================
             ! _X           ! _Y          ! _Z          ! _U          !
             !              !             !             !             !

<new skeleton> EMPLOYEE

-------------------------------------------------------------------------
! EMPLOYEE ! NAME           ! SALARY      ! MANAGER     ! DEPT        !
=========================================================================
            ! _X -          ! _Y          !             ! _Z          !
            !              !             !             !             !

<new skeleton> DEPARTMENT

-------------------------------------------
! DEPARTMENT ! DEPT        ! LOCATION     !
===========================================
              ! _Z         ! _U           !
              !            !              !

<end table definition>
<access (sub)model>

-----------------------------------------------------
! ACCESS ! USER              ! MODEL                !
=====================================================
         ! JONES             ! PAYROLL              !
         ! GEOFFRY           ! HIREDEPT             !
         ! CARTER            ! PAYROLL              !
         !                   !                      !
```

9. The report generator.

NQL is a powerful tool for report generation. The report generator enables the user to spesify the layout of his report, or let the system do the formatting. When a report is defined the user describes how the report should look and which data is going to be fetched. The description of the layout is written in a language very like the Text Formatter, and the data retrieval is described in linear NQL.
Usually report generators deal with data. We think the future user will demand more than that. We therefore want to integrate the Text Formatter of NOTIS and the report generator. We will then have a report generator doing more than the simple treatment of raw data.
This is done by writing the output from the report generator to a file. The output will contain directives to the Text Formatter. This file can then be merged into a text file using the INCLUDE command.
A report may also be written directly from the report generator to any other device.

A report may consist of one or more levels. Each level can have the following three building blocks:

heading line
: A heading at the beginning of a level. We may use information from the data base here. The heading can contain formatting directives, f.ex conditional pageshift etc.

detail line
: A detail line with information from the Data base. The detail line can continue over several lines.

footing line
: A footing at the end of a level. We can use information from the data base and format it the way we want.

The levels have to be nested. The heading lines will be written at the beginning of a new level and the footing at the end of it. The heading of the current level will be written when there is a pageshift.

9.1. Directives to define the layout.

We have some commands describing the report layout. We use the commands to describe how data is to be written.
There are two data types in the report. These are text strings and variables from the data base. A text string is a string enclosed by the aprostrophes. A data base variable is an example element. In addition we may specify the column number where the output is going to start. It is a number enclosed by paranthesis. This is optional. If the column number is omitted, the system will do the formatting.

Example:

```
 (1) 'This text starts at column one.'
(10) _VARIABLE
'The system will decide where to write this'
```

It is possible to use the following system variables in the report. They are used like other variables.

_SYSPAGE - Current page number.

_SYSDATE1 - Today's date with the format DD. <month> YYYY.
F.ex. 12.june 1979

_SYSDATE2 - Today's date with the format DD/MM/YY.
F.ex 12/06/79

_SYSDATE3 - Today's date with the format MM/DD/YY.
F.ex 06/12/79

_SYSDATE4 - Today's date with the format YY/MM/DD.
F.ex 79/06/12

_SYSTIME - The time when the report was started with the format HH:MM:SS.

Attribute values wanted in the report can be taken from the query description using example elements. If we use the example element _EX in the report layout it has to be connected to an attribute or an expression in a condition box.
We describe each of the directives we may use in the definition of the report form.
For all directives except level-up and level-down it is possible to use Text Formatter directives, such as page shift and conditional paging.

9.1.1. Level Down

^LD;

Specifies the beginning of a new level in the report. All variables used for counting on this level will be set to zero.

9.1.2. Level Up

^LU;

Ends the current level.

9.1.3. Heading Lines

^HL,string;

Defines the heading lines of the current level. String contains the commands defining the heading lines. We may let the system write the heading. If we omit the <,string> the system will write the names of the attributes mentioned in the detail line as a heading.

9.1.4. Detail Lines

^DL,string;

Defines the detail lines of the current level. String contains the commands defining the detail lines.

9.1.5. Footing Lines

^FL,string;

Defines the footing lines of the current level. String contains the commands defining the footing lines.

9.1.6. Page Heading

^PH,string;

Defines the heading of a new page. String contains the commands defining the heading.

9.1.7. Page Footing

^PF,string;

Defines the footing of a new page. String contains the commands defining the footing.

9.1.8. Comments to the directives.

In some cases it may be necessary to use imbedded commands in a command string. In this case the string should be included in 'quotes'. The start quote is ^< and the end quote is ^>.
Note that the page heading and page footing are omitted if we write the report to a text file. The Text Formatter must have directives defining these from the text file 'calling' the report text. When we write the report to any other device, the pageheading and pagefooting will be written.

Example:

```
^PH=^< (1) 'This is a report' (60) 'Page'
      (66) _SYSPAGE ^BL; (1) 'written by me ^>;

^PF=^< (1) 'This is the end' ^BL;
       (1) 'of a page' ^>;
```

will give the output:

```
This is a report                                        Page 1
written by me
.
.
.
This is the end
of a page
```

9.2. The change operator.

When we want three attribute values to be printed from records in the data base we can do it in the following way

```
^DL= _NAME    _ARTICLE    _NUMBER ;
```

The result is

```
OLSEN       CAR          3
OLSEN       CAR          4
OLSEN       CAR         12
OLSEN       BICYCLE     15
OLSEN       BICYCLE      6
WEBSTER     NUT          2
WEBSTER     WASHER      10
WEBSTER     SCREW        5
JONES       CAR          1
JONES       CAR          3
```

Sometimes we don't want to print the same attribute value many times below each other.
We want the following way of writing it

```
OLSEN       CAR          3
                         4
                        12
            BICYCLE     15
                         6
WEBSTER     NUT          2
            WASHER      10
            SCREW        5
JONES       CAR          1
                         3
```

This can be achieved with the change operator. If we specify CHANGE. before the example element containing the attribute value, the value will be written only if the new value is different from the old one.

When we want to use the operator on two or more attribute values, we have to use a priority key. The priority is a number enclosed by parenthesis.

For the above example we have to write the following detail line

```
^DL= CHANGE(1)._NAME    CHANGE(2)._ARTICLE    _NUMBER ;
```

When we have a page shift, the first line on the new page will be written with all attribute values visible.

9.3. Underlining of attribute values.

Often it is wanted to underline attribute values with a single or double underlining.
This can be specified with commands.

9.3.1. Single underline.

If we write the command SINGLELINE. in front of an example element the attribute value will be underlined with a single line.

The detail line

```
^DL= 'This is a value with a single underline' SINGLELINE._PRICE
```

will give the following print

```
This is a value with a single underline          1234,45
                                                 -------
```

9.3.2. Double underline.

If we write the command DOUBLELINE. in front of an example element the attribute value will be underlined with a double line.

The detail line

```
^DL= 'This is a value with a double underline' DOUBLELINE._PRICE
```

will give the following print

```
This is a value with a double underline          1234,45
                                                 =======
```

9.4. Examples of use

It is difficult to understand how the report generator works without any examples of its use. Two examples are shown to clarify the definition of the reports.

9.4.1. Report for a sales company.

We write a report for a company producing different articles. They want a report of what each of their customers have ordered and a total sum of all the orders.

```
<define-query>

BEGIN
   CUSTOMER(CUSTNO:ASC._XNO,NAME:_NAME,ADDRESS:_ADDR)
   ORDER(CUSTNO:_XNO,ORDERNO:_ORD,DATE:_DATE)
   ORDERLINE(CUSTNO:_XNO,ORDERNO:_ORD,ARTNAME:_ART,QTY:_QTY)
   ARTICLE(ARTNAME:_ART,PRICE:_PRICE)
   _XPRICE     = _QTY * _PRICE;
   _SUMXPRICE  = SUM(_XPRICE);
   _SUMCUST    = SUM(_SUMXPRICE);
   _GRANDSUM   = SUM(_SUMXCUST);
   REPORTFORM
      ^PH=^< (1) 'Sales report for the XXX company' (60) 'Page'
           (66) _SYSPAGE ^>;
      ^PF= (1) 'Printed on date' (27) _SYSDATE1 ;
      ^LD;
         ^FL= (1) 'Total sum for company' (60) DOUBLELINE._GRANDSUM ;
         ^LD;
            ^HL=^< (1) 'Cust.no' (10) 'Name'   (45) 'Address' ^BL; ^>;
            ^DL= (1) _XNO   (10) _NAME (45) _ADDR ;
            ^FL=^< (1) 'Sum for customer' (60) DOUBLELINE._SUMXCUST ^BL; ^>;
            ^LD;
               ^HL=^< (10) 'Orderno' (20) 'Date' ^BL; ^>;
               ^DL= (10) _ORD (20) _DATE ;
               ^FL=^< ^BL; (10) 'Sum for order' (60) DOUBLELINE._SUMXPRICE
                    ^BL; ^>;
               ^LD;
                  ^DL= (20) _ART (55) _QTY (60) _XPRICE ;
               ^LU;
            ^LU;
         ^LU;
      ^LU;
   ENDFORM
END

<end-query>
<store-query> SALES_REPORT
<execute-query> SALES_REPORT
Output-file: SALES-REPORT:TEXT
```

We have a text file with some information about the company's budget. The sales report is wanted in there, and it is 'called' in the text. Since we write the report output to a text file, the page heading and the page footing defined in the report will be omitted. The main text file will then have the definition of these.
This text file may look like this:

```
^H1=Budget report for the XXX company;
^TL=*** For internal use only ***;
^BL;The budget for next year is .......
.
.
.
.
.
^BL=3;
^IN=SALES-REPORT:TEXT;
^BL=3;As we see the sales have been .......
.
.
.
.
```

If we wanted the sales report to be written to the LINE-PRINTER with the original layout, we could have done it in the following way:

```
<execute-query> SALES_REPORT
Output file: LINE-PRINTER
```

The report produced by the Text Formatter will look like this:

Budget report for the XXX company 1

The budget for next year is .....
.
.
.
.

Cust.no	Name	Address		
101	HANK JOHNSEN	LONDON		
	Orderno Date			
	115 230879			
		BICYCLE	2	1.937,00
		BICYCLE INNER TUPE	30	367,75
		BICYCLE TYRE VALVE	100	75,85
		BICYCLE SEAT	10	357,00
	Sum for order			2.737,60
				========
	Orderno Date			
	135 211079			
		RENAULT-12 ENGINE	2	6.175,00
		WINTER TYRE	1	375,65
		FORD ESCORT ENGINE VALVE	30	900,00
		FORD ESCORT BUMPER	5	6.789,00
	Sum for order			9.240,65
				========
Sum for customer				11.978,25
				=========

Cust.no	Name	Address		
102	DAVE LURIFAX	OXFORD		
	Orderno Date			
	223 231179			
		SAFETY BELT	10	998,35
		SPARK PLUG	500	1.978,00
		RADIATOR	10	10.123,00
		THERMOSTAT	20	456,75
	Sum for order			15.737,60
				=========

*** For internal use only ***

Budget report for the XXX company 34

Orderno	Date			
345	011079			
		SAFETY BELT	4	2.175,00
		RENAULT-16 BUMPER	1	1.987,00
		RENAULT-16 SHORT ENGINE	1	3.045,00
	Sum for order			9.240,65
				========
Sum for customer				51.978,25
				=========
Total sum for firm				632.679,25
				==========

As we see the sales have been .....
.
.
.
.
.
.
.
.

*** For internal use only ***

9.4.2. Report for a plane company.

We define a report for a plane company. They want a report of all the plane routes and their departure and arrival towns. We want the system to format the report. The report is defined in the following way.

```
<define-query>

BEGIN
   FLIGHT(FLIGHTNO:_FLIGHT,DEPARTURE:_AIRP1,
          DESTINATION:_AIRP2)
   AIRPORT(CITY:_DEPT,AIRPORT:_AIRP1)
   AIRPORT(CITY:_ARRI,AIRPORT:_AIRP2)
   REPORTFORM
      ^PH=  'Report for the plane company' _SYSPAGE
      ^PF=  'Printed on date'  _SYSDATE2 ;
      ^LD;
         ^HL;
         ^DL= _FLIGHT  _DEPT  _ARRI ;
      ^LU;
   ENDFORM
END

<end-query>
<execute-query>
Name:
Output file: LINE-PRINTER
```

The report will then look like this:

```
Report for the plane company                               1

      Flightno   Departure              Destination

      BU367      OSLO                   TRONDHEIM
      BU268      TRONDHEIM              TROMSO
      SA369      OSLO                   KRISTIANSAND
      PA101      OSLO                   WASHINGTON
      PA102      OSLO                   BUENOS AIRES
      PA103      KOBENHAGEN             RIO DE JANEIRO
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      .          .                      .
      SK356      KIRKENES               HAMMERFEST

Printed on date  09/10/79
```


Report for the plane company 2

Flightno	Departure	Destination
BK123	STAVANGER	KRISTIANSAND
BK324	TRONDHEIM	LAS PALMAS
SA369	OSLO	KRISTIANSAND
SK909	OSLO	STRASBOURG
LF324	STOCKHOLM	STRASBOURG
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
SW656	KRISTIANSUND	MOLDE

Printed on date 09/10/79

10. Definition of reserved NQL words.

We have some reserved words in NQL. These are used when a query is described. In this report they are written in English.
Users who don't use English, want to use their own language. The system will allow these reserved words to be changed.

We show how the commands can be written in Norwegian.

English	Norwegian
INSERT	INNSETT
DELETE	SLETT
UPDATE	OPPDATER
PRINT	SKRIV
MINIMUM	MINIMUM
MAXIMUM	MAKSIMUM
SUM	SUMMER
COUNT	TELL
AVERAGE	SNITT
STDEV	STANDARDAVVIK
BETWEEN	MELLOM
UNIQUE	ENTYDIG
ASCENDING	STIGENDE
DESCENDING	SYNKENDE
CONDITION	BETINGELSE
BEGIN	BEGYNN
END	SLUTT
VIRTUALTABLE	VIRTUELLTABELL
ENDVIRTUAL	SLUTTVIRTUELL
INPUTFORM	INNLESEFORMAT
OUTPUTFORM	UTSKRIFTFORMAT
ENDFORM	SLUTTFORMAT
CHANGE	ENDRING
SINGLELINE	ENKELLINJE
DOUBLELINE	DOBBELTLINJE

The commands used when we enter a mode may also be specified, and the same applies for the help commands.

11. References.

(1) O.Moldekleiv, A.Stormo

Query Languages for Relational Data Base Systems.
(Written in Norwegian)

(2) O.Moldekleiv

Compiler for Query By Example.
(Written in Norwegian)

(3) O.J.Hansen

Compiler for the relational DBMS ASTRA with access via the hierarchical DBMS RA2.
(Written in Norwegian)

(4) Reports from "Summer School on Data Base Design".
Urbino, Italy, 1979

(5) Internal ND-papers.

(6) Various ASTRA(L)-papers.

(7) ADMINS/11 referance manual.

(8) M.M.Zloof

The Syntax of Query By Example.
IBM Thomas J.Watson Research Center,
Yorktown Heights.

(9) M.M.Zloof

Query By Example: A Query Language
IBM Systems Journal, no.4, 1977

(10) J.C.Thomas, J.D.Gould

A Psychological Study of Query By Example.
National Computer Conference, Proc. AFIPS,
vol 44, pp. 431-438

(11) E.F.Codd

A Relational Model of Data for Large Shared Data Banks.
CACM vol 13, no. 6, june 1970, pp. 377-387

(12) British Computer Society Query Language Group.
BCS QLG-PUB VERSN.4

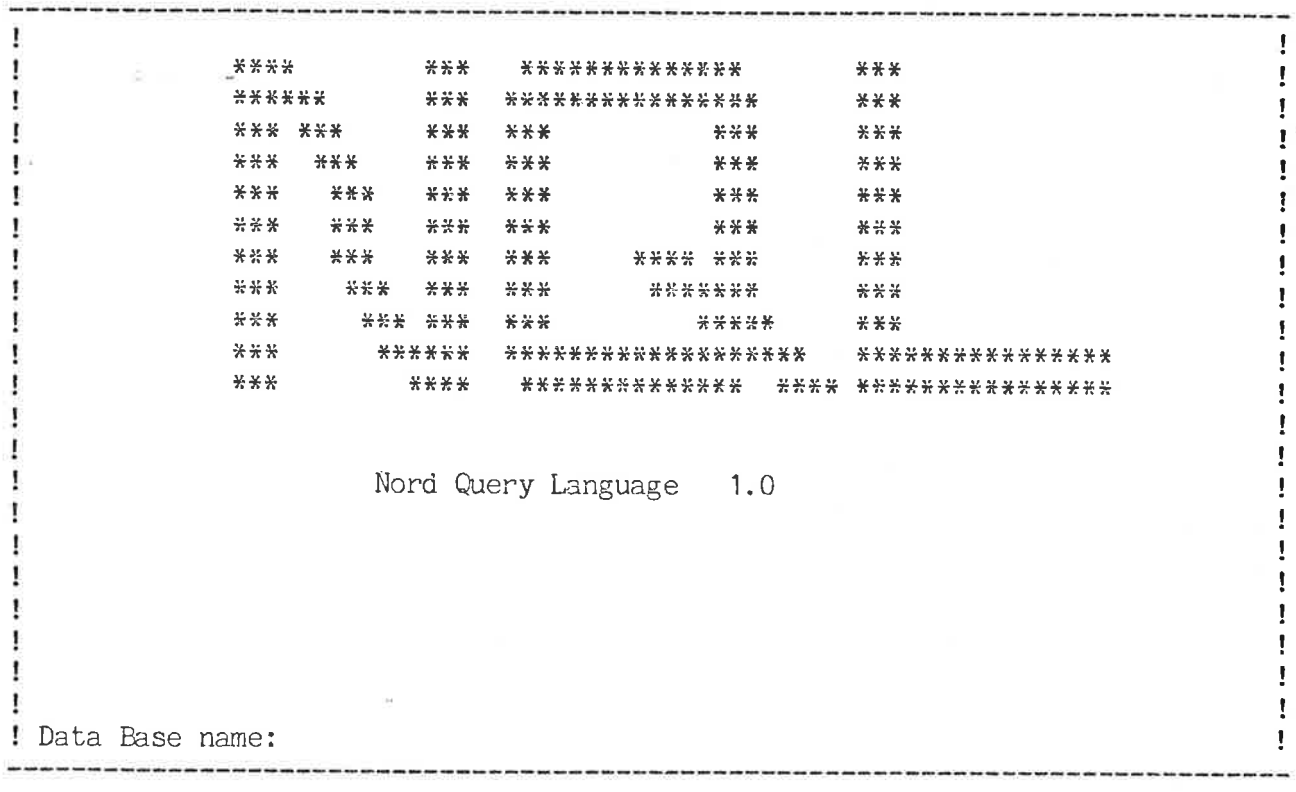
A Uniform Approach to Query Languages.

(13) CODASYL Report

A status report on the activities of the CODASYL End User Facilities Committee (EUFC)

Example of use.

The following example is taken from the author's imagination. It is only a proposal for use of the language. As the reader will see, the main layout is taken from the TED/PED editors.
When we enter the Query Language the following page will be written on the screen.
In all the example pictures, the outer frame is understood as the sceen.



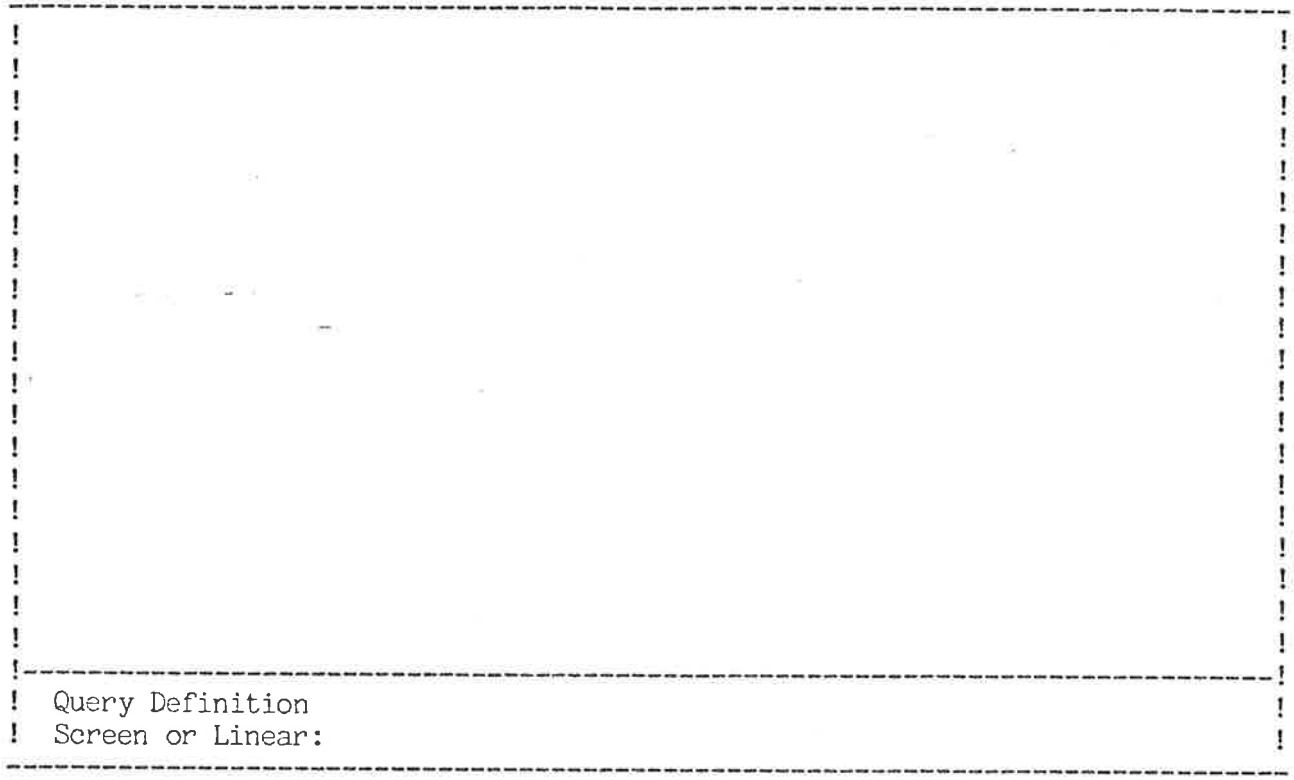
When the system is entered it will ask which data base you want to use.

It is possible to get information about the legal commands. This is achieved by typing H or ? for help.
The help command can be used when the user wants information about the data base he is working with.

```
!                                                                              !
! Nord Query Language help commands                                            !
!                                                                              !
!     Command    Function                                                      !
!                                                                              !
!     H or ?           Help mode                                               !
!     D                Query definition                                        !
!     E                Execute query                                           !
!     S                Store query                                             !
!     R                Delete query                                            !
!     I                Insert in table                                         !
!     M                Modify query                                            !
!     F                First page                                              !
!     <cursor down>    Next page                                               !
!     <cursor up>      Previous page                                           !
!     L                Last page                                               !
!     X                eXit from NQL                                           !
!     <home>           Enter command mode                                      !
!                                                                              !
!                                                                              !
!                                                                              !
!------------------------------------------------------------------------------!
! Command mode                                                                 !
!                                                                              !
--------------------------------------------------------------------------------
```

These commands can be used whenever the user wants to. The last sequence will continue when the user enters <cr>.

We want to write a query finding the name and address of all customers ordering articles with price larger than 10000.
It is done in the following example. We enter the mode for defining a query by typing D. We will get the following picture on the screen.



The system asks if we want to enter the screen definition mode or the linear definition mode.

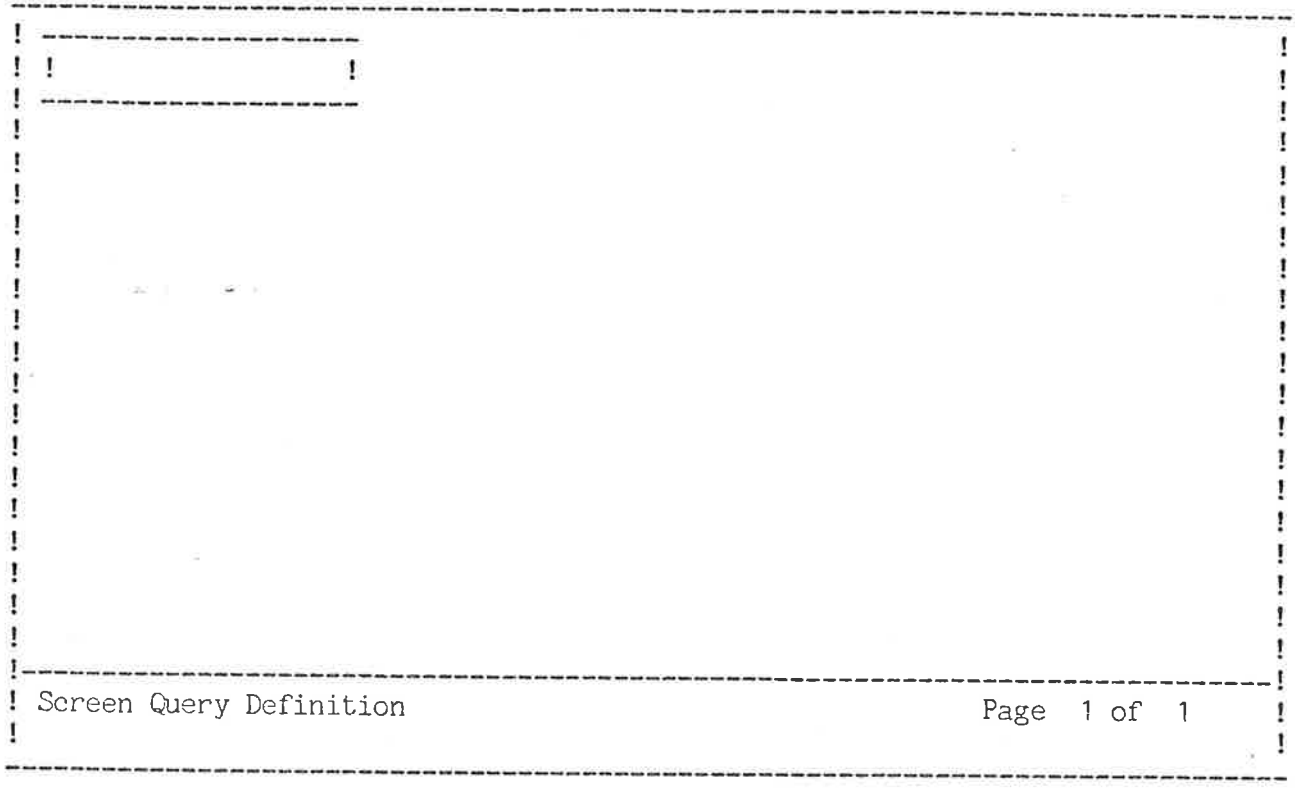
We enter S to get into the Screen Query Definition mode.

We want to see the commands available in the Screen Definition mode and enter H.

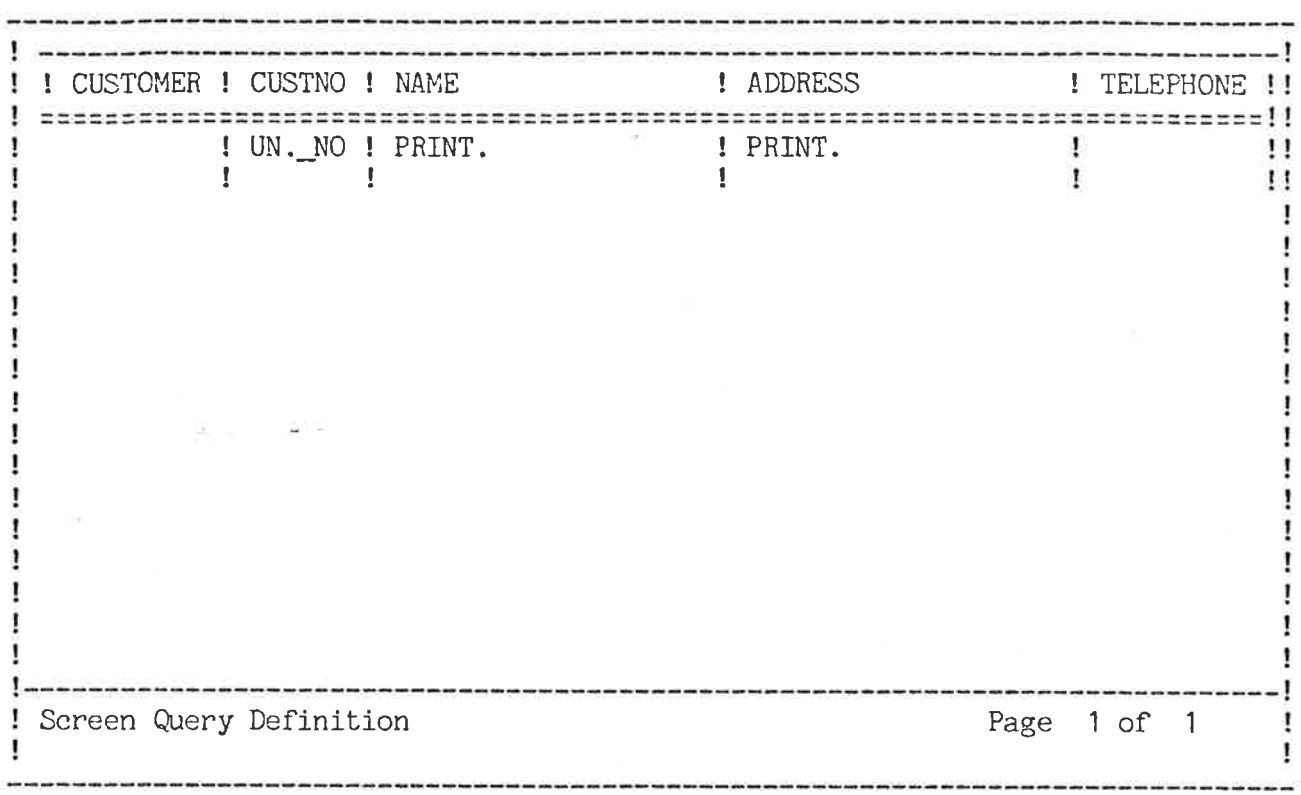
Character	HOME-mode	EDIT-mode
N	New skeleton	
V	New virtual skeleton	
F	First page	
L	Last page	
E	Exit	
H or ?	Help	
<ctrl-S>	In/Out of navigate	In/Out of navigate
<cursor up>	Previous page	Change navigate mode
<cursor down>	Next page	Change navigate mode
<cursor left>		Navigate left
<cursor right>		Navigate right
<home>	Move to last table	

Query Screen Definition Page 1 of 1

We want to enter the query commands for the first table. We want an empty skeleton of the CUSTOMER table on the screen. We enter N to get a new skeleton. The system will ask for the table name.



The cursor is positioned in the square and the system is waiting for a table name. We type CUSTOMER and the skeleton will be written on the screen. After that we can enter the query commands.


CUSTOMER
CUSTNO
NAME
ADDRESS
TELEPHONE
UN._NO
PRINT.
PRINT.
Screen Query Definition
Page 1 of 1

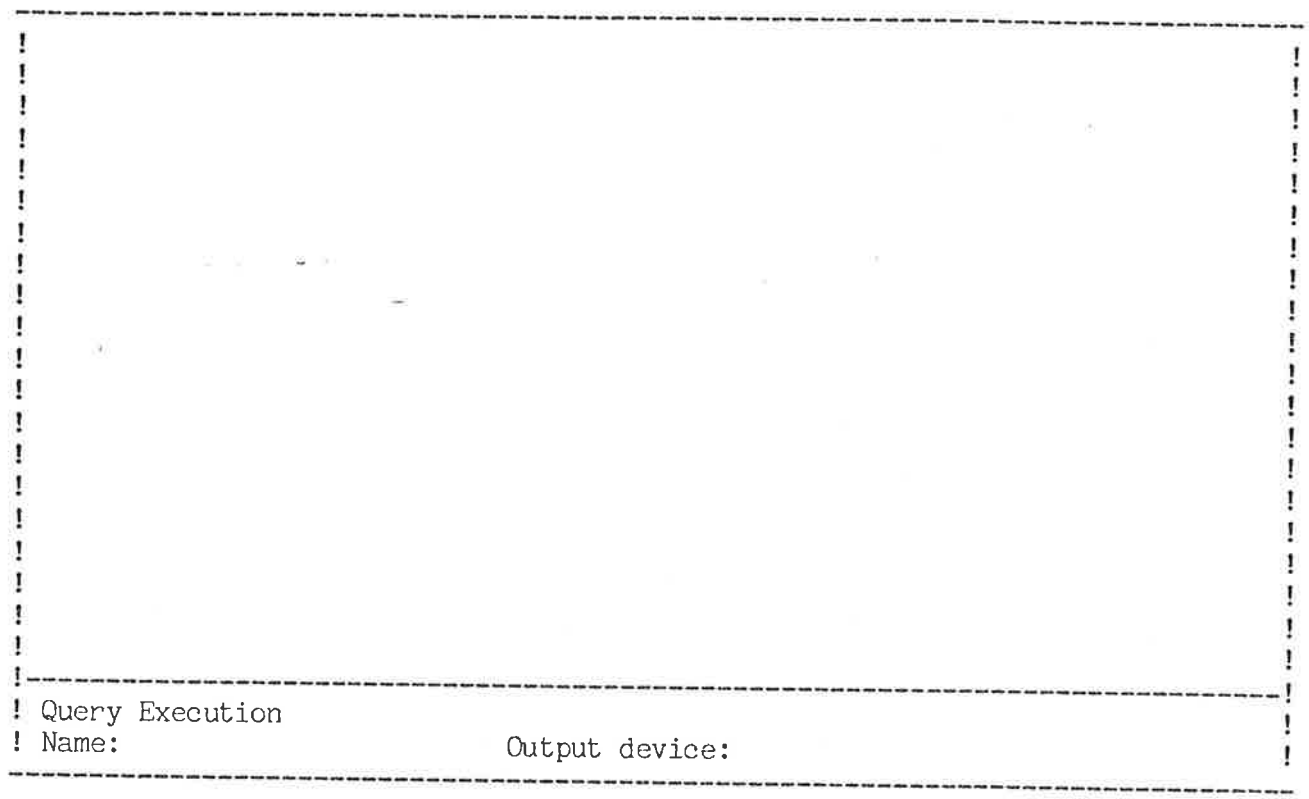
The same is done for the ORDERLINE and ARTICLE tables. We type N followed by ORDERLINE. We get the skeleton printed on the screen and enter the commands.

```
! ! CUSTOMER ! CUSTNO ! NAME                ! ADDRESS             ! TELEPHONE !!
! ==========================================================================!!
!            ! UN._NO ! PRINT.             ! PRINT.              !           !!
!            !        !                    !                     !           !!
!
!
! ! ORDERLINE ! CUSTNO ! ORDERNO ! ARTNAME            ! QTY   !
! ==========================================================
!             ! _NO    !         ! _ART               !       !
!             !        !         !                    !       !
!
!
!
!
!
!
!
!
!-------------------------------------------------------------------------!
! Screen Query Definition                                Page  1 of  1    !
!                                                                         !
```

We type N followed by ARTICLE. We get the ARTICLE skeleton printed on the screen and enter the commands.

```
! ! CUSTOMER ! CUSTNO ! NAME                ! ADDRESS             ! TELEPHONE !!
! ==========================================================================!!
!            ! UN._NO ! PRINT.             ! PRINT.              !           !!
!            !        !                    !                     !           !!
!
!
! ! ORDERLINE ! CUSTNO ! ORDERNO ! ARTNAME            ! QTY   !
! ==========================================================
!             ! _NO    !         ! _ART               !       !
!             !        !         !                    !       !
!
!
! ! ARTICLE ! ARTNAME             ! PRICE   ! STOCKQTY !
! ==================================================
!            ! _ART                ! >10000  !          !
!            !                     !         !          !
!
!-------------------------------------------------------------------------!
! Screen Query Definition                                Page  1 of  1    !
!                                                                         !
```

The query is now finished. We type E, and the system will generate code for the query. We are now in command mode. To execute the query we write E and the system will ask for the name of the query to be executed and the wanted output device. If we have a query with question elements, the system will ask for input device too.



We type <cr> two times and get the default values, the current query and TERMINAL.
The system will write the answer on the screen.

```
! ------------------------------------------------------------ !
! ! CUSTOMER ! NAME                 ! ADDRESS              ! !
! ============================================================ !
!            ! HANK WILSON          ! LONDON               ! !
!            ! CATY MILLS           ! LIVERPOOL            ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!------------------------------------------------------------!
! Query Output                                  Page  1 of  6 !
!                                                             !
---------------------------------------------------------------
```

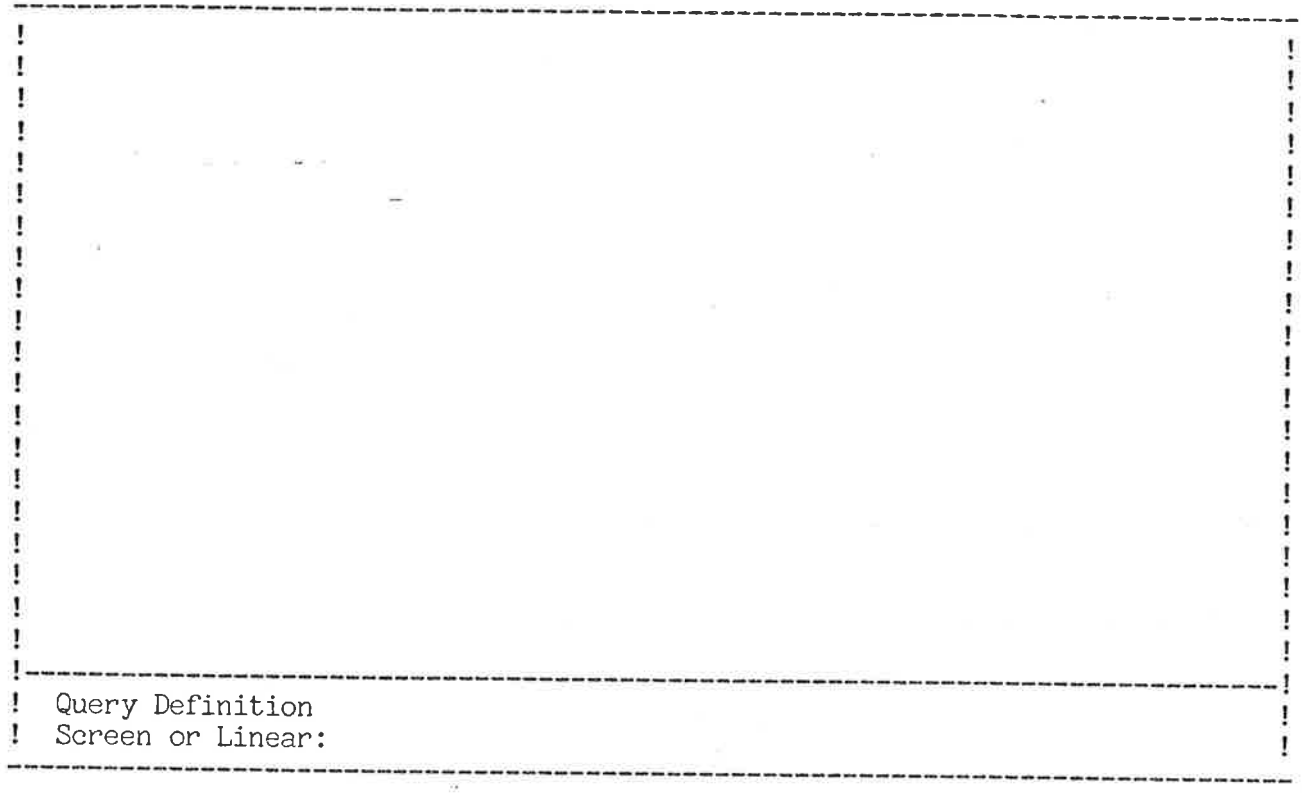
We type <cursor down> to get the next page.

```
---------------------------------------------------------------
! ------------------------------------------------------------ !
! ! CUSTOMER ! NAME                 ! ADDRESS              ! !
! ============================================================ !
!            ! GEORGE WILSON        ! MANCHESTER           ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!            ! .                    ! .                    ! !
!------------------------------------------------------------!
! Query Output                                  Page  2 of  6 !
!                                                             !
---------------------------------------------------------------
```

We type <cursor down> in order to see the next page, <cursor up> to see the previous page. F will give us the first page and L the last one.
If we have tables larger than the screen width, we have to use the horizontal scrolling commands <cursor left> and <cursor right>. In order to see all

attribute information in a table we had to scroll a page to get the wanted view.

We show an example using the linear user interface and demonstrate how user defined forms can be used.
We want a query inserting customers into the data base. We enter the mode for defining a query by typing D. The following will be written on the screen.



We type L in order to get into the Linear Query Definition mode .

We can now enter the query. We return to the command mode entering <home>.

```
BEGIN
   CUSTOMER:INPUT.(
                    CUSTNO:     ?NO,
                    NAME:       ?NAME,
                    ADDRESS:    ?ADDR,
                    TELEPHONE: ?PHONE)
   INPUTFORM
      Insertion of customer
      --------------------


      Customer number:  ?NO            Telephone: ?PHONE


           Name:        ?NAME
           Address:     ?ADDR

   ENDFORM
END
```

```
 Linear Query Definition                                    1 - 20
```

The query is now finished. We type E in order to get back to command mode. We enter E and the query will be executed. The system asks for query name and input and output file. We enter the defaults and get the following picture on the screen.

```
+-------------------------------------------------------------------------+
!                                                                         !
!       Insertion of customer                                             !
!       ---------------------                                             !
!                                                                         !
!       Customer number:  ......          Telephone: ......               !
!                                                                         !
!                                                                         !
!            Name:        ..............................                  !
!            Address:     ....................                            !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!                                                                         !
!-------------------------------------------------------------------------!
!  Query Execution                                                        !
!                                                                         !
+-------------------------------------------------------------------------+
```

We enter the values and the query will continue.

We define a report writing information about all orders a customer has. We type D and L in order to enter the Linear Query Definition mode. Then the report can be written.
The system will scroll the picture when the page is full, i.e just like the TED editor does it.

```
BEGIN
    CUSTOMER(CUSTNO:    _NO,
             NAME:      ?NAME,
             ADDRESS:   ?ADDR)
    ORDERLINE(CUSTNO:   _NO,
              ARTNAME:  _ART,
              QTY:      _ANT)
    ARTICLE(ARTNAME:    _ART,
            PRICE:      _PRICE)
    _COST  = _ANT * _PRICE;
    _TOTAL = SUM( _COST );
    REPORTFORM
       ^LD;
          ^FL= 'Total sum for all orders' DOUBLELINE._TOTAL ;
          ^LD;
             ^HL;
             ^DL= _ART _ANT _PRICE _COST ;
          ^LU;
       ^LU;
    ENDFORM
```

Linear Query Definition 1 - 20

```
    INPUTFORM
        Report for finding all
        orders belonging to a customer
        ------------------------------


        Customer name:    ?NAME
        Customer address: ?ADDR
    ENDFORM
END
```

Linear Query Definition 21 - 40

We execute the report and get.

```
!
!        Report for finding all
!        orders belonging to a customer
!        ------------------------------
!
!
!        Customer name:    ..............................
!        Customer address: ....................
!
!
!
!
!
!
!
!
!
!
!
!
!-------------------------------------------------------------------------!
!  Query Execution                                          Page  1       !
!                                                                         !
```

We enter name and address, and the query will be executed. The result will be like this.

```
!                                                                         !
!     Artname              Qty      Price         Cost                    !
!     -------              ---      -----         ----                    !
!                                                                         !
!     BICYCLE                3         769,00      2.202,00              !
!     FORD ESCORT            1      45.500,00    136.500,00              !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!     .                      .             .             .               !
!                                                                         !
!     Total sum for all orders                           546.567,35       !
!                                                        ==========       !
!-------------------------------------------------------------------------!
!  Query Output                                        Page  1  of  1     !
!                                                                         !
```

– we make bits for the future

NORSK DATA A.S BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY PHONE: 39 16 01 TELEX: 18661