INTERFACING A QUERY LANGUAGE TO A CODASYL DBMS

Roger M Tagg

Independent Consultant and Chairman of British Computer Society's
End User Systems Group (formerly Query Language Group)

(Paper originally presented at Pergamon Infotech State of the
Art Tutorial, London, December 1982)

INTRODUCTION

Many organisations' data nowadays resides in databases. In spite of a rather 'steam technology' image, CODASYL methods are used for a large number of these databases - and the number is still increasing. The need to support querying of these databases has not been addressed by the CODASYL committees, with the result that software suppliers have introduced a very diverse set of solutions to the problem of interfacing a query language to a DBMS.

Suppliers, and sometimes user DP groups, have often been under pressure to bring in quickly a query facility for a DBMS which may have been in use for some time. This pressure has sometimes meant that there has been less time to plan a good interface. Some of the current approaches, for example, have tried to evolve a query system from an interactive form of the CODASYL COBOL DML, or have attempted to bolt on an interactive report generator, or even a look-alike relational syntax.

In looking at all the current approaches, the key question seems to be: what view of data in the database should users of the query language be offered? At one extreme, should he or she have to think in terms of the record/set/realm structure in which the CODASYL subschema is expressed? Or at the other extreme, should he simply be required to know the names of the data items he wants and leave the system to put the best construction it can on what the query then means?

This paper will look at some of the current approaches and attempt to evaluate them against certain criteria. It will go on to discuss whether some approaches are preferable to others and if so, whether there is any chance of evolution of existing languages towards a more consistent, common approach.

BRIEF DETAILS OF SOME EXISTING LANGUAGES

ACCESS is a query system offered by Norsk Data which aims to interface to both CODASYL DBMS (SIBAS) and conventional or indexed files. It follows the approach of Zloof's Query-by-Example, and is discussed in another paper at this forum.

AIDA (sometimes alias GENIE) is a query language oriented to the ICL 2900 version of IDMS. It was developed by a user (Mullen & Co) rathern than ICL themselves, but with the recent appearance of ICL's QUERYMASTER it may not be a contender in the future. In the normal mode AIDA is an 'interactive DML' type of language. The user sees the whole schema and writes commands:

- FIND to indicate an entry point record type (using a key)
- SCAN to search the member records or a set or to search a record type sequentially
- GET to access data from the owner record of a set.

There is also a prompted 'question-and-answer' mode, and users can also be offered the use of macro queries, the user only being required to supply parameters.

CQLF (CODASYL Query Language Flat) is a new approach commissioned by the US National Bureau of Standards from the Computer Corporation of America (CCA), but it is not yet a product which is on the market. CQLF aims to bring a relational style to the query language. However, it is not relational in the sense of restricting user views to flat files: instead the relational calculus syntax has been extended to include operations involving sets. User views - which are effectively subschema networks - are derived from each other in a 'closed' system as with normal relational theory. The user therefore has to be aware of the network structure, and he uses this in a syntax which is designed to be similar to IBM's SQL.

DATATRIEVE is the development of DEC's DATATRIEVE-11 system into a query language for both RMS (multiple index files) and CODASYL DBMS on the VAX systems. It has not been studied in detail for this paper, but it has interesting facilities for use in a distributed environment.

FOCUS is a self-contained data management and reporting system marketed by Information Builders Incorporated. It also has a number of interfaces allowing it to retrieve data from databases such as IMS (DL/I), TOTAL, ADABAS and IDMS, the last of these being a CODASYL system. When using FOCUS as a query language for IDMS the user has to think in terms of FOCUS data structures - these are segment hierarchies (like DL/I) which can be 'rehung' to provide inverted views. Although the actual data is in IDMS, the user is unaware of the IDMS structure. Only when a totally new view is required does the user (or database administrator) have to set up a new FOCUS description and the correspondence rules stating how the IDMS structure is to be traversed. Pansophic's EASYTRIEVE also has an IDMS interface with a somewhat similar approach.

HARVEST is the query language that goes with IDBS's SEED system - a portable, FORTRAN-based CODASYL DBMS that works on a variety of hardware, just as TOTAL or RAPPORT do. It attempts to offer the user an 'independent item' of data, working out the path through the database itself. However if this process proves ambiguous or impossible, the user may have to help the system by identifying which record type is to be treated as the TARGET record, ie the record type at which 'occurrences' satisfying the query should be counted. This process clearly requires some understanding of the network structure.

IQL is a query language for the DEC10 and DEC20 DBMS, which is of CODASYL type, but again has not been investigated for this paper.

MANAGE QUERY is the language for CSC's MANAGE CODASYL DBMS, which CSC offer over their INFONET time-sharing service. The user's view is based on 'structures'. A structure is a simplified network defined within a subschema. In a user's query he can simply refer to individual items as long as they exist within the current structure. In cases of ambiguity he can identify the BASE record (like HARVEST's TARGET record) using a BASED ON clause. He can switch from one structure to another, but if a new structure is required a subschema compilation is necessary. Thus naive users can take an 'independent item' view within a structure, while more experienced users can use some knowledge of the network data structure.

MDBS-QRS is a query and reporting language which interfaces to MDBS, a CODASYL DBMS sold on a variety of micros and minis which use the CP/M (or MP/M), Unix or RSTS operating systems. Suppliers are Micro Data Base Systems Inc. This language has not been studied in detail for this paper.

OLQ2 is offered by Cullinane as their query language for the IBM version of IDMS. It has developed from an interactive DML, but a PATH concept has been introduced. Within a defined path the user need only refer to named data items, but a path must be essentially linear. A new path can be defined within the query language, but to do this the interactive DML form of the language has to be used. However this combination does provide a gradation of views for both naive and experienced users.

PRESENT is a fairly recent language which operates with Data General's CODASYL DBMS. Like OLQ2 it uses a PATH mechanism. However, if the user does not know about paths, and just supplies names of data items, PRESENT will attempt to find an existing path that will support the query. Also like OLQ2 the path definition mechanism is outside the subschema mechanism - it uses a special PRPATH utility.

PRIME DBMS QUERY is also of relatively recent release. Here basic views are termed VIRTUAL RECORDS. These are structures (similar to MANAGE) which are defined within the subschema; however some VIRTUAL RECORDS are automatically generated by subschema compilation - these consist of each record type plus all the data from its owner record types in the various sets, and their owners, etc. In the query statement the user has to identify the virtual record to be used in a FROM clause, though the 'automatic' virtual records have the same name as their base DBMS records.

QLP is one of the best-established CODASYL DBMS query languages. It operates with Univac's DMS system. Again linear paths are used, but they are defined within the subschema. However if only independent data items are named in the user's query QLP will find a path, if one exists.

QRP is also well-established, operating with Honeywell's IDS-II DBMS. The equivalent of a path in QRP is an ENTRY, which is defined within a special, subschema-like description known as an Application Definition File (ADF). Again, the system will attempt to choose an ENTRY if none is specified.

QUERYMASTER is the new query language for ICL's IDMS (and also conventional files) and is discussed in another paper. QUERY VIEWS are simplified structures set up within the subschema, so are more akin to MANAGE structures than to QLP or OLQ2 paths. However, unlike MANAGE, the CODASYL SET details are still available to the user. A user may still be able to put an 'independent items' query, but if this is ambiguous, or he wants to control the access path to be used, he has to specify a path within the query view by commands such as STARTING WITH, WITH Set-name, or linking conditions (for a relational-type join).

QUERY/UPDATE has a long history as CDC's query language for the DMS-170 DBMS - in fact it was originally a stand-alone system. More recently, it has been re-vamped for the second-generation IMF DBMS, which is CODASYL-based but has the full 3-level architecture. The user's view is similar to AIDA: a query has to specify a path which names SETS to be searched and entry point keys for OWNERS or CALC records.

Besides the systems described above, two other proposed approaches are worth mentioning. One is the approach of the CODASYL End-User Facilities Committee, perhaps the group which might have proposed a 'standard' query language. This group's work has perhaps been more forward-looking than this, and revolves around a model of user/computer interaction where all activity is seen in terms of a conceptual automated office. The basic view of data this group proposes is that of a FORM. All data is assumed to be envisaged as if it appeared on 2-dimensional displays eg pages of a report, VDU screens etc. This committee's proposals for a FORM description language will be referred to again later **(001)**.

Finally, the report of the BCS Query Language Group proposed the generalised concept of a PERCEIVED RECORD **(002)**. This is seen as the collection of data relating to objects (entities) of interest to the user rather than being tied to the database structure. For example, most existing query languages operating on a CUSTOMER --> ORDER --> ORDER-ITEM hierarchy would nominate ORDER-ITEM as the base record of any path, even though the user might be interested in finding out about the customers who have placed orders for certain things. In the PERCEIVED RECORD approach, the natural hierarchy would be visible as well as the collapsed-down path. The natural hierarchy would, in general, include the possibility of repeating groups.

To summarise, there is clearly some evidence of convergence of ideas towards the provision of some sort of predefined view for use in the query language. But the way these views are implemented seems to differ in some respects in

almost every query language looked at. And despite the convergence, there is the possibility that these views are not general enough to support the natural concepts of the users, such as forms and entities.

COMPARISON OF APPROACHES

The different routes taken by each approach can be compared under three main groups of criteria:

- How the user's view of data is supported
- How the user operates his query procedure
- General usability and support features.

An additional consideration is the extensibility of the language to operate on data from a different DBMS or conventional files. In Figure 1, under the user's view group, the questions to be considered are:

1 What does the user need to know?

   - At the simplest level
   - At more expert levels.

2 Can a view based on named data items only be supported?

3 Can a 'perceived record' be referenced?

   - A simple linear path
   - A more general structure with a 'base record'
   - A more general structure with a hierarchy of repeating groups.

4 How can an experienced user create new perceived record definitions?

5 How do user views relate to CODASYL subschemata?

Within the query procedure group, the questions to be considered are:

6 What mode of dialogue is offered?

   - User command driven
   - Prompt-driven (question and answer)
   - Other.

7 Can the user progressively refine the scope of a query?

8 Can formatted reports with sorting and control breaks be produced?

9 Does the system support limited updating?

- o Temporary updates for 'what if' queries
- o Permanent changes to the database.

10 Can the user display results in graphical form?

11 Can macros of query statements be created for later reference and use?

- o Full queries with parameters solicited at run-time
- o Substrings of queries for shorthand or other purposes.

In the general usability and support group, the questions to be considered are:

12 Can the user obtain information about the data available to him for querying?

- o Items and records within user views/perceived records
- o The underlying CODASYL subschema structure
- o Previously defined macros.

13 Is there a HELP facility?

- o Tutorials for each command
- o HELP 'in context' of current query
- o Recap of session so far.

14 Are session control facilities provided?

- o Access control and privacy
- o Saving of selected data
- o Warning of long searches.

Finally (question 15), can the language access data other than in the CODASYL DBMS in separate queries and within the same query, joining CODASYL record types to external files?

Many of these questions are general evaluation criteria for any query language. However, several of these, in particular questions 1-5, 12 and 15, are particularly relevant to the case of CODASYL DBMS. Figure 1 shows, where the information has been readily obtainable, how some of the query languages introduced in the last section meet the various criteria above.

Question	AIDA	CQLF	FOCUS	HARVEST	MANAGE
1 User needs to know					
a) Simple usage	Subschema	Items, records, sets	Items	Items	Items, structures
b) More expert	Subschema	Items, records, sets, schema	Items, FOCUS segments	Items, record types	Items, records, structures
2 Independent item view	NO	NO	Must name a FOCUS file	YES	Must name a structure
3 Defined perceived rec.					
a) Simple linear path	Only	All using derived sub-	YES	Automatic	YES
b) Structure	Via	derived	YES	unless	YES
c) Hierarchy with RG	Macros	subschemata	NO	ambiguity	YES
4 How to create new perceived rec's		Derive new subschema (within CQLF)	Define new FOCUS structure		Choose new base record or COMPILE subschema
5 Reference to subschemata	Direct	Direct	Indirect	Direct	Indirect
6 Dialogue mode					
a) Command	YES	YES	YES	YES	YES
b) Prompt	YES	NO	NO	NO	NO
c) Other	NO	NO	NO	NO	NO
7 Progressive refine	(Edit macros)	(Derived subschemata)	(Edit query or save files)	(Edit query)	(Edit query or save files)
8 Report	YES	NO	YES (V.G.)	YES	YES
9 Updating					
a) Temporary	NO	NO	YES	YES	NO
b) Permanent	NO	YES	YES	NO	YES
10 Graphics	NO	NO	YES	NO	NO
11 Macros					
a) Whole procs	YES	NO	YES	YES	YES
b) Substrings	YES	NO	YES	NO	NO
12 Dictionary info.					
a) User views	YES	YES	YES	YES	(Structure chart)
b) Subschema	YES	YES	NO	YES	NO
c) Macros	YES	NO	NO	NO	NO
13 HELP					
a) Command	YES	NO	NO	YES	NO
b) Context	YES	NO	NO	NO	NO
c) Recap	NO	NO	NO	NO	NO
14 Session control					
a) Access control	YES	via subschemata	YES	YES	YES
b) Save files	NO	NO	YES	NO	YES
c) Timing warnings	NO	NO	NO	NO	NO
15 Other files					
a) Separately	NO	NO	IMS,VSAM,TOTAL,ADABAS etc	NO	ALADIN (INFONET files
b) Within a query	NO	NO	NO	NO	NO

Figure 1: Evaluation criteria for query languages

OLQ2	PRESENT	PRIME	QLP	QRP	QUERYMASTER	QUERY/UPDATE
Items, path Items,records,sets	Items Items, path	Items,virtual records Items,records,sets	Items Items,records,sets	Items Items,records,sets	Items, views Items,records,sets	Items, sets Items, sets
Must name a path	YES	Must name a virtual record	YES	YES	Must name view	NO
YES (by backtracking) NO	YES YES NO	YES YES NO	YES NO NO	YES YES NO	YES YES NO	Path specified within query
'Define path' command	'PRPATH' utility	Compile subschema	Compile subschema	Create an ADF description	Compile subschema	
Direct	Indirect	Indirect	Indirect	Indirect	Indirect	Direct
YES NO NO	YES NO NO	YES NO NO	YES NO NO	YES YES NO	YES NO NO	YES NO NO
YES	(Edit query or macro)	(Edit query)	(Edit query)	(Edit query)	(Edit query or save hit file)	(Edit query)
YES	YES	YES	YES	YES	YES	YES
NO NO	NO NO	NO NO	NO YES	NO NO	NO NO	NO YES
NO	YES	NO	NO	NO	NO	NO
YES NO	YES NO	YES YES	YES NO	YES NO	YES YES	? ?
YES YES YES	YES NO YES	YES YES YES	NO NO NO	YES NO YES	YES NO YES	? ? ?
YES NO NO	YES NO NO	YES NO YES	NO NO NO	YES NO NO	YES NO NO	? ? ?
YES NO NO	YES NO NO	YES NO NO	Via subschemata (COBOL) NO NO	YES NO NO	YES YES NO	? ? ?
VSAM NO	NO NO	(In future) NO	COBOL files NO	NO NO	YES YES	NO NO

DISCUSSION OF MAIN ISSUES RAISED BY THE COMPARISON

The primary issue, as has already been emphasised, is in the way that user views of data are offered and defined. Questions 1 to 5 illustrate the difference in approach between the current languages. The approaches can be grouped as follows:

1 'Automatic' generation of the user's view based on data items named from the full subschema (HARVEST).

2 'Automatic' generation of the user's view from data items selected from named substructures defined within the subschema (MANAGE, QUERYMASTER).

3 Selecting the user's view from one of a number of predefined path structures defined within the subschema (QLP).

4 As 3 above, but path structures defined outside the subschema (QRP, PRESENT, FOCUS).

5 Using as the user's view a specified path structure defined within the subschema (PRIME).

6 As 5 above, but path structures defined outside the subschema (OLQ2).

7 Full navigation within the subschema (AIDA, CQLF, QUERY/UPDATE (and OLQ2 without PATHS)).

The first few approaches obviously require less knowledge on the user's part, but have to cope with the possibility of the resulting query not being what the user really wants. The later ones require the user to know more details about the data structure, but having specified the query in more detail there is less chance of misunderstanding.

Leaving aside the approaches in point 7 above - they are more suitable for very experienced users, programmers and database administrators - there are possibly two basic levels of user interface for the data view, both of which should ideally be provided by the query system:

- Automatic, purely based on items in the query
- Oriented towards a specific 'perceived record' named by the user.

The option in 3 and 4 above, of selecting a particular perceived record from those already defined could also be provided, but it is really a 'variation' between these two levels, and could be handled through dialogue management. The same applies to choice of the base record in level 1.

The choice of whether perceived records are defined inside or outside the subschema seems to be a cosmetic issue as far as normal users are concerned. In most of the approaches 1 to 6 reference to the subschema is indirect, ie users only reference constructs local to the query language. The issue becomes important when new perceived records have to be created - this is

assumed to be done by database administrators or expert users rather than casual or inexperienced users.

It seems on balance preferable that the definition mechanism for new perceived records should be oriented towards the use within the query sessions rather than restricted to database administrators. With many DBMS compilation of subschemata is a specialist task, and this may have encouraged the use of external definitions (eg QRP, PRESENT, OLQ2). In some other systems on-line subschema compilation utilities have been introduced, partly in answer to this need.

Whichever method is chosen, the result is still essentially a subschema facility, ie a mapping of the data available from the schema. This packaging question is addressed again in the final section of this paper. Figure 2 below shows diagrammatically some of the possible mappings involved in supporting the user's view of data.

A Query System Subschema would contain descriptions of data items available to the user, and perceived records available for reference. The term 'Perceived Records' is assumed to cover Virtual Records, Paths, Entries, Structures and Query Views as used in current systems. It could be, as just discussed, separate from or merged with the normal CODASYL subschema.

Another important issue, already raised earlier, is the generality of the perceived record structures available (question 3, Figure 1). Some current languages are restricted to a linear path, but others allow any subset of the subschema network structure. In between these two, some offer linear paths but with automatic availability of any data from owner records (and their owners etc). Current systems do not generally support a user view which consists of an owner record with all its members treated as a repeating group (possibly nested), as envisaged as one of the options for a perceived record in the BCS Query Language Group report. In fact, despite the fact that CODASYL allows repeating groups <u>within</u> records, only MANAGE and CQLF offer any syntax to support this.

The problem about the use of general subnetworks as a basis for user views is that they still require a further stage of processing before an access path can be determined. The two chief examples of this, MANAGE Structures and QUERYMASTER Query Views, are perhaps best regarded as equivalent to whole Query System Subschemata - similar to HARVEST where the whole subschema is the basis for user views.

It seems more appropriate to use an extended PATH construct as the perceived record basis. Such paths would have a <u>base</u> record type defined, and data from owners (as unique occurrences), and from members (as repeating groups), would be supported. Figure 3 shows how some alternative perceived records could be defined from the same source schema.

Another major issue concerns the provision of information to the user about the data available to be queried, ie 'dictionary' data. Most current languages recognise the need to support this function - some will even draw up Bachman diagrams. What is less clear is whether the dictionary data for the query system should reside in the main data dictionary, or be kept in files specific to the query system.

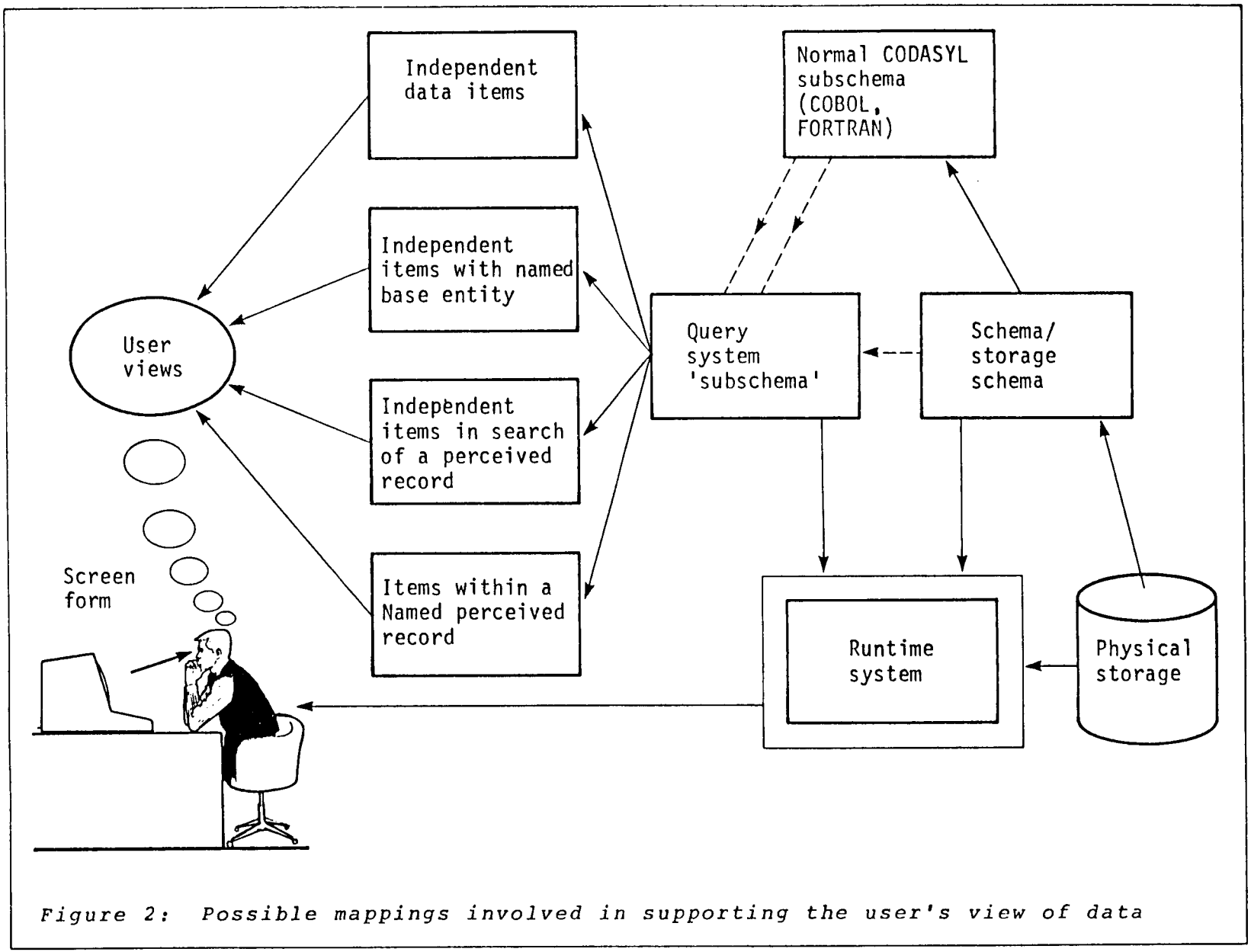

Figure 2: Possible mappings involved in supporting the user's view of data

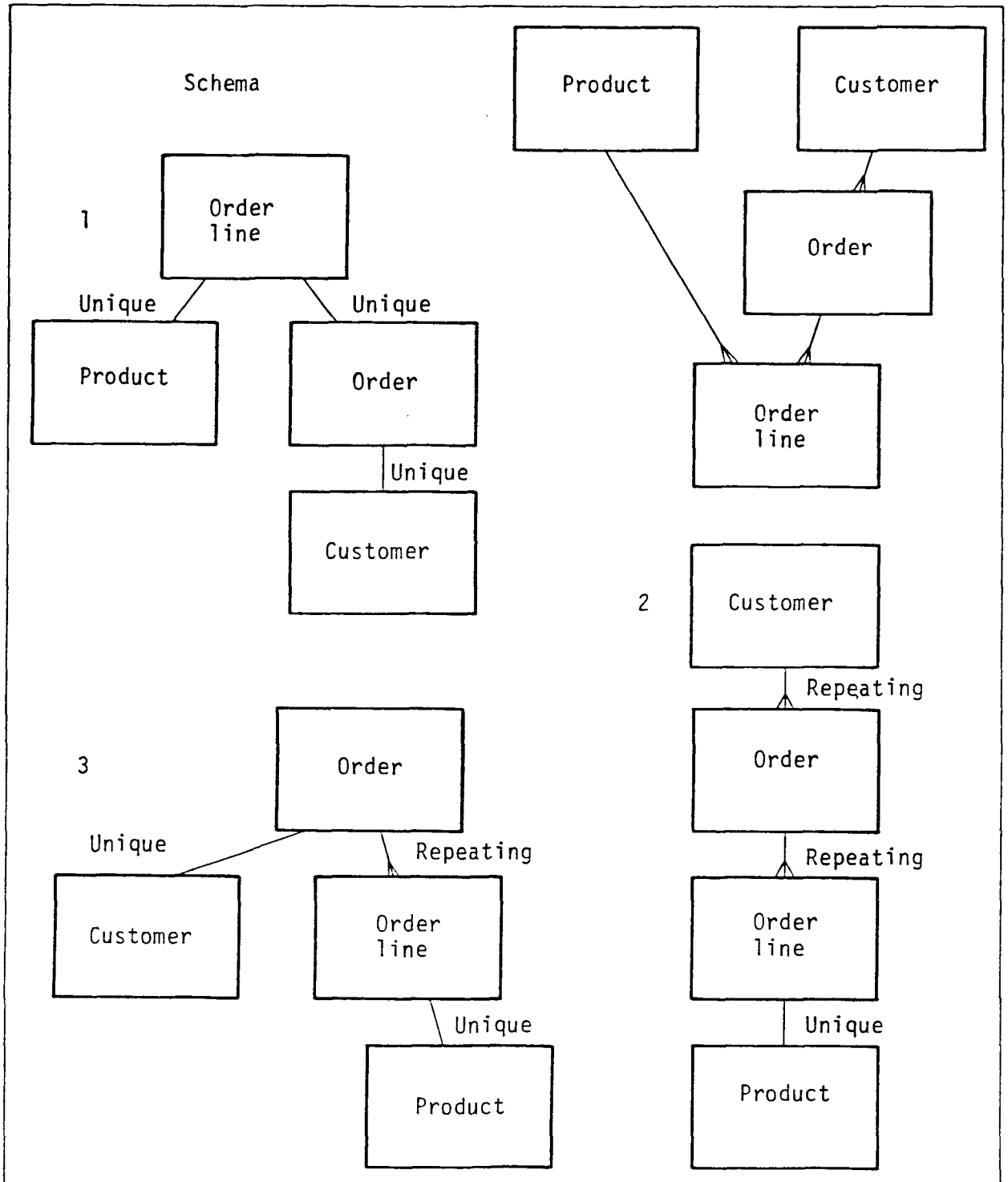

Figure 3: Defining alternative perceived records from the same source schema

If the dictionary is integrated with the DBMS, and in fact uses the DBMS for its own structure, it would seem sensible to handle query language descriptions just like any other dictionary data.

Dictionary queries can then be supplied to the users either by query macros or genuine ad-hoc requests. If a stand-alone dictionary is used, this may be a problem unless the query language can also access data in the dictionary-s file organisation: if not, a separate dictionary in CODASYL DBMS form would be better, otherwise special software would be needed for dictionary queries.

On this last point (see question 15, Figure 1), only a few of the current languages allow queries on non-DBMS files, and only QUERYMASTER can mix DBMS and non-DBMS data within a single query. FOCUS demonstrates that it is possible to have one language that can interface a wide range of DBMS and file types, and it is developing the ability to mix these in a query. It seems that this is a desirable aim, and this therefore means that the query system subschema facility should be separated from the normal CODASYL subschemata used for COBOL, FORTRAN etc.

The remaining discussion covers issues other than those concerned with user views of data.

In question 6, Figure 1, virtually all the current languages are command oriented - only AIDA and QRP have built-in prompted modes. However in most of the others tailored prompted dialogues can be set up by a database administrator or expert user. ACCESS (not in the comparison) is unique in its QUERY-BY-EXAMPLE mode.

The results of question 7, Figure 1, show a general weakness in the field of providing the user with any facilities for gradually refining his query in a 'research' mode. If he asks a query which will output 10 000 lines and he wants to add an extra condition, then the usual route is to re-execute the query suitably edited. Only OLQ2 (with GET DBKEYLIST) and QUERYMASTER and MANAGE (with recycable saved files) avoid this.

Reporting (question 8) is more variable than the comparisons suggest. Updating (question 9) is more the exception than the rule. Temporary updating seems a worthwhile aim for 'what if' queries while permanent updating involves defining of transaction start and end points to ensure consistency. Direct graphics in the query language (question 10) is only provided in FOCUS and PRESENT - this seems a very desirable extension for a query language.

Most systems allow macros of whole query procedures which can be executed later, with parameters supplied at runtime if desired. Macros of substrings are less common, but this is a very useful way of allowing users to tailor dialogues to their own preferences using abbreviated forms, special terminology and even foreign languages (see question 11).

The need for HELP (question 12) and session control is recognised, but coverage is very variable in existing languages, with no facilities for warning about a potentially long query execution.

POTENTIAL PROPOSAL FOR A NEW SUBSCHEMA FACILITY

The discussions in this paper have led towards some ideas of how a future, more powerful and possibly standardised query language interface might look. The remainder of this paper outlines a possible proposal that could be put to the CODASYL Data Description Language Committee (DDLC) to introduce a new Query Subschema facility into the CODASYL specifications.

At a meeting of this committee (DDLC) in September 1981 the following 'straw votes' were decisively supported:

1 The area of proposing subschema language facilities for a query language interface to CODASYL DBMS should be a responsibility of DDLC.

2 The query subschema should support perceived records which are materialised from multiple schema record types.

3 The query subschema should support multiple perceived records which may draw information from the same schema record.

4 A subschema with the facilities desired for a query language could also be useful for host languages (COBOL etc).

It has also been felt, as a result of many discussions, that the user's query view of data should not normally involve knowledge about SETS in the CODASYL database. This is partly a corollary of the conclusion of the BCS Query Language Group report **(002)** , where query users were assumed to view a single perceived record type at a time. Another argument is that if the query language should interface to <u>any</u> file management system or DBMS, a construct used only in one type of DBMS (eg a CODASYL SET) is not desirable.

Three particular references are available which show how this objective of a SET-free query view can be provided:

- The FOCUS/IDMS interface **(003)**
- The Cullinane IDMS Version 5.7 Logical Record Facility **(004)**
- The CODASYL End User Facilities Committee's EUFFDL **(005)** (End User Facility Data Description Language).

The FOCUS/IDMS interface is defined in two parts, the Master file and the Access file. The Master file is the normal description of data as if it were stored in FOCUS files - hence it is independent of the CODASYL database structure. The ACCESS file describes a path through the IDMS database which supports the virtual views described in the Master file.

A Cullinane Logical Record is also defined in two parts, but both parts are within the normal subschema description. The <u>definition</u> part shows only which physical (schema) record types are included within the Logical Record type. The second part is a PATH GROUP statement. This allows specification of one or more paths which support access to the Logical Record type. The reason for having several such paths is to support efficient processing of

```
FORM ENTRY

    SOURCE SECTION        DBMS SCHEMA NAME IS scheme-name

    NAME SECTION          FORM NAME IS form-name

    ITEM SECTION          ITEM NAME IS item-name
                             [PICTURE IS ---] [USAGE IS ---]
                             [SOURCE IS ---] [DESCRIPTION IS ---]

    STRUCTURE SECTION ROOT IS group-name-1 KEY IS ---
                          group-name-2 CONTAINS { group-name-3 } [REPEATS]
                                                { item-name    } [ORDER IS ---]

    DATA SECTION          SEARCH entry-point-record-type [GIVING ---]
                              THEN member-record-type [BY set-name] [GIVING ---]
                              OWNER owner-record-type [BY set-name] [GIVING ---]
                              FROM branch-point-record-type
                              EXAMINE RELATIONSHIP bomp-owner-record-type
                                THEN bomp-member-record-type BY set name-1
                                OWNER bomp-owner-record-type BY set name-2
                                GIVING ---  UP to integer TIME[S]
                              LOOKUP record-type (not involving a SET)
                                WHERE linking-condition [GIVING ---]

ITEM/GROUP
LIMITER SECTION           EXCLUDE { item-name  } WHEN condition
                                  { group-name }

GEOMETRY SECTION          (detailed screen  or print layout of FORM).

FILE ENTRY

    NAME SECTION          FILE NAME IS file-name

    FORM SECTION          FORM NAME IS form-name  [ORDER IS ---]

    FORM LIMITER
       SECTION            EXCLUDE FORM WHEN condition.
```

the Logical Record type from a variety of anticipated entry points (eg updating, serial search or retrieval on specific values or ranges of various key fields). However for ad hoc access, optimisation of such paths by the system would be preferable.

In the EUFDDL a variety of clauses is used to describe each 'FORM' (ie end user perceived data view). An additional concept is a 'FILE', which is a set of occurrences of FORMS according to some selection criteria. Several FILES could be defined on the same FORM. The clauses in the EUFDDL are as shown on the preceding page.

The EUFDDL seems to offer a very powerful set of options for perceived record structuring. It allows hierarchies with repeats as well as the normal PATH structures of most CODASYL Query Languages. It even allows iterative access of a BOMP structure. However the syntax shows some features which could raise questions:

1 All mapping is done through items. The ITEM SECTION names all items on the FORM as if independent. The structure section forms them into groups for the query view, and the GIVING clauses identify the source items that are extracted from the schema records. Some means of mapping across whole records from schema to query view groups might be preferable in many cases.

2 Mapping functions are split between the DATA SECTION, LIMITER SECTIONS and ITEM SECTION. The DATA SECTION does the hard work of defining the access path and naming the items to be extracted. The LIMITER SECTION defines which items and groups (or whole forms in the FILE ENTRY) are to be excluded. The ITEM SECTION defines, in the SOURCE clause, the means of deriving the FORM data items from the data items in the GIVING clauses. From the packaging and DBMS-independence viewpoint, it would seem preferable to keep all CODASYL-dependent mapping declarations in a separable part of the description.

3 Data inherent in a CODASYL SET (set order position, count of current members) is not able to be materialised onto a FORM.

The syntax suggested for consideration by CODASYL DDLC therefore, is a modified form of the CODASYL EUFDDL which aims to solve these problems:

```
      DESCRIPTION SECTION
2         PERCEIVED RECORD NAME IS perceived-record-name-1
             ROOT SEGMENT IS segment-name-1
3               CONTAINS {item-name-1 [description-clauses]} ...
4            [ ( SEGMENT segment-name-2 [REPEAT] WITHIN { Segment-name-3 } ) ] ...
                                                        { ROOT          }
3               CONTAINS  item-name-2 [description-clauses] ...
```

```
     MAPPING SECTION
5         SOURCE IS  ||  DBMS SCHEMA schema-name                    ||
                     ||{EXTERNAL FILE file-name-1} ...               ||
6         MAPPING FOR PERCEIVED RECORD perceived-record-name-2
               SEARCH {dbms-record-name-1                  }
                      {EXTERNAL file-name-2                }[WHERE condition] [GIVING clause]
                      {PERCEIVED perceived-record-name-3   }
          [  ||  THEN  dbms-record-name-2 [BY set-name-1] [WHERE condition] [GIVING clause]  ||  ]
             ||  OWNER dbms-record-name-3 [BY set-name-2] [WHERE condition] [GIVING clause]  ||
             ||  FROM dbms-record-name-4                                                     ||
             ||  EXAMINE RELATIONSHIP dbms-record-name-5 [THEN clause] [OWNER clause]        ||
             ||                                          [UP to integer-1 TIME[S]]           ||
             ||  LOOKUP {dbms-record-name-6 } WHERE linking-condition [GIVING clause]        ||
             ||         {EXTERNAL  file-name-3}                                              ||
```

Details of the GIVING clause are as follows:

```
7   GIVING {[SEGMENT segment-name-4]
8          {ITEM[S] {item-name-3 AS {{RENAME  }
9                                   {TOTAL   }
                                    {AVERAGE } schema-item-name-1
                                    {MAX     }
                                    {MIN     }
                                    {SET-COUNT   }
                                    {            }set-name-3
                                    {SET-POSITION}
                                    RESULT OF PROCEDURE procedure-name ON
                                                        {schema-item-name-2} ...
                    } } ... } ...
```

Notes on this syntax are as follows:

1 The query subschema is a collection of perceived records, and the data items within them, available to a user within a single query section. The user is assumed either to name a single perceived record type at a time, or to express queries solely in terms of data items in the query subschema (in which case the query system will attempt to construct an implicit, temporary perceived record type).

2 For each perceived record type defined a 'root segment' is defined, containing data items which occur once in each occurrence of the primary entity type which the user is assumed to be considering.

3 All items available to the user within a segment are named as they are to be referred to in queries. Names can be the same as schema names if desired. Description clauses include USAGE and PICTURE but not SOURCE: defaults imply no change from the schema.

4 Different segments generally correspond to different entities or schema record-type subsets. However they can be used to separate repeating groups within a schema record type. Repeats can be specified as subordinate either to the root segment directly, or to another named segment.

5 There is just one SOURCE clause for the MAPPING SECTION. The clause has been placed here to concentrate the DBMS dependence within the MAPPING SECTION. Reference to non-DBMS files is catered for in the syntax.

6 The mapping for each perceived record type will contain one SEARCH clause (for the entry-point record-type in the DBMS, the prime external file, or a define perceived record-type). Subsequently there may be a number of THEN, OWNER, FROM, EXAMINE AND LOOKUP clauses, using the same meanings as for the EUFDDL.

7 The GIVING clause specifies how query data items are derived from record types on the path. In the absence of any clause, query data item names are assumed to match the schema names and direct mapping by name will take place. If only SEGMENT is quoted, then any query data names in the named segment synonymous with data item names in the current schema record type, will be matched. If ITEMS are included, then the mapping for each individual query item can be specified.

8 RENAME refers to a schema data item in the current DBMS record type. TOTAL, AVERAGE, MAX and MIN all refer to an item either within a repeat in the current DBMS record type, or in a member record with respect to some SET in the access path. SET-COUNT works similarly to these functions for a named SET-POSITION (ie the sequence number within the SET order) will form a query item within the segment representing the member record type of the SET. The PROCEDURE option will specify its own access route within the procedure itself, but could cover data from members, owners or lookup records.

9 Segments derived from items in a schema record repeating group can be 'normalised out' if all the items are within the repeat.

10 A GEOMETRY SECTION is not included as display and report layouts are assumed to be ad hoc in a query facility.

11 A FILE ENTRY is not included as the subsets of perceived records which are to be defined will again be mainly ad hoc in a query language. The WHERE clauses for SEARCH, THEN and OWNER allow an initial sub-setting to be defined.

REFERENCES

001 CODASYL End-User Facilities Committee J of Development
CODASYL Canadian Federal Government (1980)

002 British Computer Society Query Language Group
Query languages - a unified approach
Heyden Press/BCS Monographics in Informatics (1981)

003 FOCUS User Manual IDMS
Information Builders Inc (1981)

004 IDMS Release 5.7 Features Guide
(see Sections on Logical record facility)
Cullinane Database Systems Inc (1981)