# NASA Contractor Report 172191

NASA-CR-172191
19830025312

# DBPQL: A VIEW-ORIENTED QUERY LANGUAGE FOR THE INTEL DATA BASE PROCESSOR

Paul A. Fishwick

Kentron Technical Center
Hampton, Virginia 23666

Contract NAS1-16000
July 1983



National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

## SUMMARY

An interactive query language (DBPQL) for the Intel Data Base Processor (DBP) is defined. DBPQL includes a parser generator package which permits the analyst to easily create and manipulate the query statement syntax and semantics. The prototype language, DBPQL, includes trace and performance commands to aid the analyst when implementing new commands and analyzing the execution characteristics of the DBP. The DBPQL grammar file and associated key procedures are included as an appendix to this report.


## INTRODUCTION

DBPQL (Data Base Processor Query Language) is the third and final layer of the HILDA system. HILDA ("High Level Data Abstraction System") is a three-layer data base management system supporting the data abstraction features of the Intel Data Base Processor (DBP). The purpose of HILDA is the establishment of a flexible method for efficiently communicating with the Intel Data Base Processor. Each layer within HILDA plays a specific role during this communication. These roles may be seen in figures 1 and 2. Figure 1 displays the method by which one may flexibly modify the syntax and semantics for the data base machine. Figure 2 shows the anatomy of a sample query made to the data base processor. The first layer within HILDA is SPP (Service Port Protocol, ref. 1) and serves as the underlying data communications protocol allowing full access to the DBP data base management functionality. The second layer of HILDA represents a procedure package which contains command encoding procedures. The procedure package is termed "DBPSSP" (Data Base Processor Semantics Specification Package, ref. 2) and enables the analyst to easily create request blocks to be transmitted to the DBP. DBPQL was built using PARGEN (ref. 3) and other tools available within the MYSTRO system. DBPQL is designed to be utilized by both casual data management users and users who have an in-depth knowledge of the DBP commands.

## The DBPQL/DBP CONCEPTUAL COMMAND DICHOTOMY

The DBP Reference Manual contains an in-depth description of the format for the request and response modules. The manual also includes a BNF-type "conceptual"

command scheme which relates to the DBMS functions using a one-to-one mapping. That is, each internal conceptual DBP command may be conceptually defined by an external user-oriented BNF syntax production. The conceptual command is similar to a machine instruction on a conventional machine in that it represents the lowest, indivisible level of DBP functionality.

DBPQL was designed with the idea that the vast majority of data base machine users are not interested in exploring the functionality at the level described in the reference manual. This implies that there is a definite one-to-many mapping of DBPQL query commands to Intel DBP conceptual commands. The semantics afforded by each conceptual DBP command are encapsulated within the file "DBPCMD.DAT" which is included as Appendix A. This file includes a set of procedures which are called by the semantics within the DBPQL grammar file. Each procedure represents a single conceptual DBP command with the arguments necessary to build the request block portion relating to that command.

A strong attempt was made to shield the casual user of the data base machine from abstruse and often confusing functions such as the following:

1. FREE - free a currently attached view from the session.
2. ATTACH - attach a view to the user's session.
3. KEEP - make an entity (database, file, view) permanent.
4. SUBMIT KEYS - submit keys to the session's keyring.
5. DEFINE FILE - define a file within a given database.

As an example of the complexity involved with using certain DBP conceptual commands, we consider the DEFINE FILE command. Whenever it is necessary to create a relation, one must first define a "file." In defining a file one must specify page size, volume identification, initial allocation size and maximum allocation size.

These functions may be quite useful to the user experienced with the DBP, but they should be transparent to a user who simply wishes to easily and quickly manipulate data. Most of the functions such as those listed above are required during data base management queries. In order to make them transparent, their functionality is woven into the semantic definitions of the relevant queries.

The data within the user's database is manipulated through relational algebra which is performed on views (ref. 4). Views in DBPQL are defined to be windows which map onto the general set of data, allowing the user to see only the relevant data sections within the database. Views may be easily created from other views via relational commands such as the following:

```
create project view CONNECTIVITIES from QUADS
    including NODE1 NODE2 NODE3 NODE4
```

In this example, the user wishes to see only the connectivity information present within the relation "QUADS." This information is extracted from items "NODE1," "NODE2," "NODE3," and "NODE4" and a new view named "CONNECTIVITIES" is created in the process.

The experienced user of the data base processor is accommodated through the use of "options." Options permit the user to be specific about certain database and relation creation parameters such as internal page size, variable item area size within fields, and allocation sizes. The adjustment of these parameters provide flexibility for experienced users who wish to fine tune their DBP databases and files. Using options, a user may wish to specify explicit links or other performance-enhancing measures in his database. The inexperienced user can simply assume the defaults in most cases and not be unduly affected.

In the following sections within this report, the method which is used to develop DBPQL will be shown. This same method may be applied in other research efforts to develop an entirely different high-level user interface. The first section describes PARGEN, a programming tool used to develop the syntax and semantics of DBPQL.

## PARGEN

PARGEN stands for PARser GENerator and is a program contained within the MYSTRO (ref. 3) system developed at the College of William and Mary. Many of the terms and concepts introduced in this section may be found in any compiler design textbook such as reference 5. As its name implies, PARGEN is used to easily

generate compilers and query processors that contain embedded parsers. PARGEN expects two inputs before it can execute:

1. Grammar File - contains the syntax and semantics for the language to be generated. The syntax is specified in terms of BNF (Backus-Naur Form) productions. The semantics are written in Pascal directly following the syntax production to which they are related.

2. Skeleton Compiler or Query Processor - contains a minimal language compiler/query processor which has embedded tags to aid PARGEN in the correct insertion of certain variables and the synthesize case statement. The case statement is used during the parsing phase to activate the semantics associated with a specific rule being fired.

As output, PARGEN produces the new compiler/query processor which contains everything necessary to correctly parse the user's particular source program. PARGEN also produces the parse tables which are used by the compiler/query processor during the parsing phase.

It should be noted that PARGEN can handle certain grammar ambiguities such as shift-reduce and reduce-reduce conflicts which occur regularly when designing new languages. In addition, productions may contain semantic conditions which must be true for the production to be applied. These semantic conditions may be used to resolve a given reduce-reduce conflict in the user's grammar. The class of grammars that can be handled by PARGEN is the NQLALR(1) type (ref. 6).

The execution of PARGEN may best be portrayed with an example. An arithmetic expression grammar is shown as an example of the grammar input file to PARGEN:

```
?ALL
?CRUSHER
?ERC
<GOAL> ::= <EXPR> <EOLN>
  WRITELN('THE ANSWER IS = ',SSTACK[MP].IVAL);
<EXPR> ::= <EXPR> + <FULL_TERM>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL + SSTACK[SP].IVAL;
<EXPR> ::= <EXPR> - <FULL_TERM>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL - SSTACK[SP].IVAL;
<EXPR> ::= <FULL_TERM>
 ;
<FULL_TERM> ::= <TERM>
 ;
<FULL_TERM> ::= + <TERM>
 ;
<FULL_TERM> ::= - <TERM>
  SSTACK[MP].IVAL := -SSTACK[SP].IVAL;
<TERM> ::= <TERM> * <FACTOR>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL * SSTACK[SP].IVAL;
<TERM> ::= <FACTOR>
 ;
<TERM> ::= <TERM> / <FACTOR>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL DIV SSTACK[SP].IVAL;
<FACTOR> ::= <PRIMARY> ** <PRIMARY>
 ;
<FACTOR> ::= <PRIMARY>
  SSTACK[MP].IVAL := SSTACK[SP].IVAL;
<PRIMARY> ::= <NO>
 ;
<PRIMARY> ::= ( <EXPR> )
  SSTACK[MP] := SSTACK[MP+1];
```

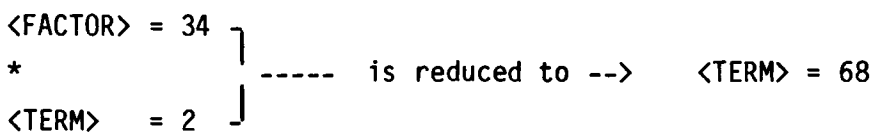
The semantic text is able to "pick off" the appropriate command/source line tokens by accessing the semantics stack which is maintained in the compiler. The semantics stack variable "SSTACK" may be referenced as follows:

Suppose that the user types in the expression: 2*34

One production that would fire during the parsing of this expression would be:

```
<TERM> ::= <TERM> * <FACTOR>
 SSTACK[MP].IVAL := SSTACK[MP].IVAL *SSTACK[SP].IVAL;
```

Note the semantics for this production rule. Three items are expected on the top of the stack:




Through the semantics, the three stack items are replaced by the result of the multiplication. The variable "MP" refers to the left-hand side production symbol (LHS), and the variable "SP" refers to the last token in the right-hand side (RHS):

| | | | |
|---|---|---|---|
| <FACTOR> | SSTACK[SP] | or | SSTACK[MP+2] |
| * | SSTACK[SP-1] | or | SSTACK[MP+1] |
| <TERM> | SSTACK[SP-2] | or | SSTACK[MP] |

The above expression grammar, when run through PARGEN, will produce an expression evaluator program. The evaluator will ask the user for a given arithmetic expression, and then produce the result. Note that this rather compact grammar can handle quite sophisticated input such as the following:

```
-> 2*(3+4)/(3**( 3*(4+1) ) + 1)
```

The order of operator precedence is contained within the proper "parsing order" inherent within the syntax productions.

For further in-depth information on PARGEN, reference the PARGEN User's Manual (ref. 3). In the reference manual there are several other options which have not been mentioned here.

## AN OVERVIEW OF DBPQL

DBPQL (Data Base Processor Query Language) has been designed and developed with the aid of PARGEN. The primary purpose of DBPQL is to allow the user a simple and flexible access tool in communicating with the Intel Data Base Processor. Since DBPQL is intimately related to PARGEN, a system developer maintains the

flexibility to easily create an entirely new query grammar or modify the existing one. As the needs of the data base machine users change, the developer may change and adapt the query processing language accordingly.

DBPQL is entirely "view-oriented," as the title of this paper suggests. This means that all data to be placed into or retrieved from the database is done via a view. The entire procedure necessary to work with DBPQL may be best described using a sequence of steps:

1. Create a database - The database will hold the data which is to be defined and transformed later.

2. Create a relation - A relation is similar to a table with a set of rows (tuples) and columns (attributes). The relation identifies an underlying table which represents the "structure" within the database. There may be many relations within a single database. When one creates a relation, an "identity" view is immediately assigned for that relation. When one is "looking" at the identity view, one is viewing the entire relation as originally defined.

3. Create a view - A user will inevitably wish to look at the relation(s) in a different way than they were originally defined. Relational algebra is used (the syntax of which will be defined later) to aid in viewing the data differently. Through the use of relational algebra the user may create views upon other previously defined views. The identity view is considered to be the base view upon which all other views are constructed. Data may be henceforth retrieved and stored by directly accessing an appropriate view.

4. Display a view - Display a given view in tabular form.

5. Input from a file - Take all subsequent command input from a file containing a sequence of DBPQL commands.

6. Load a view - Load data into a given view. It should be noted that data may be loaded not only into the identity view, but also into a view that was created from other views.

7. List information - Provide a list of information about views and databases.

8. Trace - Trace the encoding and decoding of blocks to/from the DBP.

9. Performance Monitoring - Measure the performance of given DBP operations.

10. Delete a view - Delete a view that is no longer needed.

11. Delete a relation - Delete a relation that is no longer needed.

12. Delete a database - Delete a database that is no longer needed.

A view is analogous to a window (either in the real world or as in computer graphics). One is actually sectioning off a particular part of the world of data (or database) and using this modular new section for further communications.

## THE DBPQL GRAMMAR

DBPQL is a context-free query language which is represented in the BNF form specified by PARGEN. The complete grammar file is included in this report as Appendix B, however, a more concise form is listed below (without semantics):

1. <CREATE_DATABASE> ::= CREATE DATABASE <DBNAME>
2. <CREATE_RELATION> ::= CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA <SCHEMA> <OPTIONS>
3. <CREATE_CONNECT_VIEW> ::= CREATE CONNECT VIEW <NEW_VIEW> FROM <SOURCE_VIEW1> <STRING_PTR> <SOURCE_VIEW2>
4. <CREATE_JOIN_VIEW> ::= CREATE JOIN VIEW <NEW_VIEW> FROM <SOURCE_VIEW1) <ITEM1> <SOURCE_VIEW2> <ITEM2>
5. <CREATE_ORDER_VIEW> ::= CREATE ORDER VIEW <NEW_VIEW> FROM <SOURCE_VIEW> <ITEMS...> <DIRECTION>
6. <CREATE_PROJECT_VIEW> ::= CREATE PROJECT VIEW <NEW_VIEW> FROM <SOURCE_VIEW> [INCLUDING | EXCLUDING] <ITEMS...>
7. <CREATE_SELECT_VIEW> ::= CREATE SELECT VIEW <NEW_VIEW> WHERE <WHERE_CLAUSE> <OPTIONS>

8. <DELETE_VIEW> ::= DELETE VIEW <VIEW>
9. <DELETE_RELATION> ::= DELETE RELATION <RELATION>
10. <DELETE_DATABASE> ::= DELETE DATABASE <DATABASE>
11. <DISPLAY> ::= DISPLAY <VIEW>
12. <HELP> ::= HELP [ <DBPQL_COMMAND> ]
13. <INPUT> ::= INPUT
14. <LISTDB> ::= LISTDB <DATABASE> | ALL
15. <LISTDBS> ::= LISTDBS
16. <LIST_VIEW> ::= LISTVIEW <VIEW>
17. <LIST_VIEWS> ::= LISTVIEWS
18. <LOAD> ::= LOAD <VIEW> <ITEMS_TO_LOAD>
19. <PERFORMANCE_COMMAND> ::= PERFON | PERFOFF
20. <TRACE_COMMAND> ::= TRACEON | TRACEOFF

Once a view has been created, the user may either display the view (using DISPLAY) or load data into it (using LOAD). The structure of the database and views may be shown using the LISTVIEW, LISTVIEWS, LISTDB, and LISTDBS commands. Appendix C contains an actual DBPQL/user dialog during the creation of a finite-element model database. Also shown in Appendix C is the function of the (TRACEON,TRACEOFF) commands which permit an optional display of the request and response modules that are being transmitted between the host and the data base machine. The trace commands enable the developer to easily verify the command encodings and the proper interpretation of the DBP responses.

A HELP command was added to aid the user in using DBPQL. Typing HELP <CR> at the terminal will yield a list of all available commands. To obtain a description of any one command, one must type HELP <command>.

## A General Form of a DBPQL Query Statement

When referencing the DBPQL grammar file in Appendix B, one will notice a consistent structure in the formation of the syntax productions. This general structure is shown below:

<QUERY> ::= <KEYWORD> <QUERY_REST> <OPTIONS>

1. Start the encoding of the request module.
2. Remove the command line tokens from the symbol tables and call the appropriate conceptual command procedures.
3. Send the request module to the DBP.
4. Process the response module.

<KEYWORD> ::= XXXXXX

1. Initialize counter variables.
2. Set all defaults now.

<QUERY_REST> ::= <QUERY_DEPENDENT_ARGUMENTS>

1. Take the tokens from the semantics stack and store them in the appropriate symbol tables.

<OPTIONS> ::= <OPTIONAL_CLAUSES>

1. Usually involves setting flag variables which override the default settings.

The inclusion of the "<OPTIONS>" production allows the experienced user to tailor a specific database environment to his needs. On the other hand, the casual user is not forced to supply the system with complicated details, since the details are optional.

## CONCLUDING REMARKS

The syntax of DBPQL is not unusual. There are many examples of query languages whose syntax closely resembles the DBPQL syntax. The unusual aspect of the DBPQL development resides with the use of two concepts which will be discussed in the following paragraphs.

The first concept is the parser generator. The parser generator, PARGEN, used in forming DBPQL is considered to be an integral, embedded part of the DBPQL system. PARGEN is not to be used solely by the developer of the initial query language. PARGEN is designed to accompany DBPQL (or another language) if DBPQL is distributed so that the end-users have a choice in modifying or enhancing the grammar to suit their local needs. Most currently available data base management packages allow no changes in their syntax and semantics since these packages contain "built-in" parsers. The DBPQL research has shown the advantage of having the parser generator and the query language together as a packaged system.

The second concept is that of a rigorous semantics specification. At first, the DBPSSP semantic procedures were used directly in the grammar file. Then, after designing several syntax productions, it became evident that there was a cleaner method of accomplishing the task of coding the semantics. The meaning (or semantics) of each DBP conceptual command is captured in a one-to-one relationship with a "conceptual procedure." The conceptual procedures contain DBPSSP semantic procedures, while the grammar file contains calls to the conceptual procedures. Implementing the conceptual procedures seemed to make the task of preparing a grammar file a simple one. Also, since many different queries will contain the same semantics (such as to attach and free session views), the grammar file is more compact and comprehensible.

In developing the grammar file for DBPQL, it was annoying to constantly have to invent new variables which act as symbol tables during parsing. This meant that it was necessary to modify the skeleton query processor to insert the variable declarations. Perhaps the symbol tables ought to reside on the DBP side inside a symbol table relation. This would mean slower data access to the DBP-resident symbol tables, but the independence of the grammar file and skeleton query processor would be facilitated. In other words, one would not have to modify the skeleton processor to include the variable definitions for symbol table storage and manipulation.

## REFERENCES


1. Fishwick, Paul A.: SPP: A Data Base Processor Data Communications Protocol. NASA CR-172144, May 1983.

2. Fishwick, Paul A.: DBPSSP: A Data Base Processor Semantics Specification Package. NASA CR-172172, June 1983.

3. Noonan, Robert E.; and Collins, Robert: The MYSTRO Parser Generator PARGEN User's Manual: Version 6.2. College of William and Mary, August 1982.

4. Adiba, M.: Derived Relations: A Unified Mechanism for Views, Snapshots, and Distributed Data. Proceedings of the Seventh International Conference on Very Large Data Bases, Cannes, France, Sept. 1981.

5. Gries, David: Compiler Construction for Digital Computers. John Wiley & Sons, Inc., 1971.

6. DeRemer, Frank; and Pennello, Thomas J.: Efficient Computation of LALR(1) Look-ahead Sets. Proceedings of SIGPLAN Symposium on Compiler Construction, pp. 176-187, Aug. 1979.

# APPENDIX A - DBPQL Conceptual Procedures

```
$ type [intel.dbpql]dbpcmd.dat

(* SUPPORT PROCEDURES FOR DBPQL - EXTERNAL *)

PROCEDURE INIT; FORTRAN;
PROCEDURE START; FORTRAN;
PROCEDURE TRON; FORTRAN;
PROCEDURE TROFF; FORTRAN;
PROCEDURE BITSB; FORTRAN;
PROCEDURE BITS( BYTEVALUE:INTEGER ); FORTRAN;
PROCEDURE BITSE; FORTRAN;
PROCEDURE ASC( %STDESCR STRING:PACKED ARRAY[INTEGER]
               OF CHAR; LENGTH:INTEGER ); FORTRAN ;
PROCEDURE ASCX( %STDESCR STRING:PACKED ARRAY[INTEGER]
                OF CHAR; LENGTH:INTEGER ); FORTRAN;

PROCEDURE INT1( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT2( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT4( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE TERMINATE; FORTRAN;
PROCEDURE SEND; FORTRAN;
PROCEDURE RECV( VAR RESPONSE:LIMIT;
                VAR TOTAL_BYTES:INTEGER;
                VAR MORE:BOOLEAN ); FORTRAN;
PROCEDURE PERFON; FORTRAN;
PROCEDURE PERFOFF; FORTRAN;
PROCEDURE TRACEON; FORTRAN;
PROCEDURE TRACEOFF; FORTRAN;

(* UTILITY PROCEDURES *)

PROCEDURE NUM_TO_ASCII( NUMBER:INTEGER;VAR ASCII_NUMBER:IDENT_STRING;
                        VAR ASCII_NUMBERL:INTEGER );

(* CONVERT AN INTEGER TO ASCII *)

VAR  COUNT    : INTEGER;
     COUNT2   : INTEGER;
     DIGIT    : INTEGER;
     WORKING_NUMBER: INTEGER;
     STRING   : IDENT_STRING;

BEGIN
   WORKING_NUMBER := NUMBER;
   COUNT := 0;
   REPEAT
     COUNT := COUNT + 1;
     DIGIT := WORKING_NUMBER - ( WORKING_NUMBER DIV 10 )*10;
     WORKING_NUMBER := WORKING_NUMBER DIV 10;
     STRING[ COUNT ] := CHR( DIGIT + 48 );
   UNTIL WORKING_NUMBER = 0;
   (* REVERSE THE DIGITS *)
   FOR COUNT2 := 1 TO COUNT DO
      ASCII_NUMBER[ COUNT2 ] := STRING[ COUNT - COUNT2 + 1 ];
   ASCII_NUMBERL := COUNT
END;
```

```
(* RECEIVE 'DESCRIBE VIEW' RESPONSE *)

PROCEDURE DV_RESPONSE( VAR VIEW:IDENT_STRING;VAR VIEWL:INTEGER;
          VAR VIEW2:IDENT_STRING;VAR VIEW2L:INTEGER;
          VAR NUM_ITEMS:INTEGER;
          VAR ITEM_NAME:IDTYPE;VAR ITEM_NAMEL:NUMTYPE;
          VAR ITEM_TYPE:NUMTYPE;VAR ITEM_LENGTH:NUMTYPE );


VAR     COUNT :INTEGER;
        COUNT2:INTEGER;
        OFFSET:INTEGER;

BEGIN
   OFFSET := 16;
   VIEWL := ORD( RESPONSE[ OFFSET ]);
   VIEW := BLANK_IDENT;
   FOR COUNT := 1 TO VIEWL DO
       VIEW[ COUNT ] := RESPONSE[OFFSET+COUNT];
   (* POINT OFFSET TO 'VIEW-OWNER' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   VIEW2L := ORD( RESPONSE[ OFFSET ]);
   VIEW2 := BLANK_IDENT;
   FOR COUNT := 1 TO VIEW2L DO
       VIEW2[ COUNT ] := RESPONSE[OFFSET+COUNT];
   (* POINT OFFSET TO 'READ-LOCK' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'WRITE-LOCK' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'READ-WRITE' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'FILE-COUNT' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;

   NUM_ITEMS := ORD(RESPONSE[OFFSET+3]);
   OFFSET := OFFSET + 5;
   FOR COUNT := 1 TO NUM_ITEMS DO
   BEGIN
      ITEM_TYPE[ COUNT ] := ORD(RESPONSE[OFFSET+2]);
      ITEM_LENGTH[ COUNT ] := ORD(RESPONSE[OFFSET+3]);
      ITEM_NAMEL[ COUNT ] := ORD(RESPONSE[OFFSET+4]);
      ITEM_NAME[ COUNT ] := BLANK_IDENT;
      FOR COUNT2 := 1 TO ITEM_NAMEL[ COUNT ] DO
          ITEM_NAME[ COUNT,COUNT2 ] := RESPONSE[ OFFSET+COUNT2+4];
      (* UPDATE OFFSET TO THE NEXT ITEM *)

      OFFSET := OFFSET + ITEM_NAMEL[ COUNT ] + 5;
      OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1;
      OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1
   END
END;



(* QUERY *)

PROCEDURE QUERY;


TYPE  HEXTYPE = ARRAY[1..2] OF CHAR;
```

```
VAR   NUM_ITEMS: INTEGER;
      COUNT    : INTEGER;
      COUNT2   : INTEGER;
      COUNTER  : INTEGER;
      COUNTER2 : INTEGER;
      OFFSET   : INTEGER;
      ITEM_COUNT : INTEGER;
      DBCOUNT : INTEGER;
      VIEWCOUNT : INTEGER;
      FILECOUNT : INTEGER;
      SPACE_FILL : INTEGER;
      HEXSTRING : HEXTYPE;


PROCEDURE TOHEX( BYTE_VALUE:CHAR; VAR HEXDIGITS:HEXTYPE );

  FUNCTION HEXDIGIT( NUMBER:INTEGER ):CHAR;

  BEGIN
     CASE NUMBER OF

     0,1,2,3,4,5,6,7,8,9: HEXDIGIT := CHR( NUMBER+48 );

     10 : HEXDIGIT := 'A';
     11 : HEXDIGIT := 'B';
     12 : HEXDIGIT := 'C';
     13 : HEXDIGIT := 'D';
     14 : HEXDIGIT := 'E';
     15 : HEXDIGIT := 'F'

     END
  END;


BEGIN
   HEXDIGITS[1] := HEXDIGIT( ORD( BYTE_VALUE ) DIV 16  );
   HEXDIGITS[2] := HEXDIGIT( ORD( BYTE_VALUE ) MOD 16  )
END;


BEGIN
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );
   IF TOTAL_BYTES = 0 THEN
      WRITELN('Ok')
   ELSE
   BEGIN
      CASE ORD( RESPONSE[ 1 ] ) OF

      (* FETCH RESPONSE *)

      (* F1 *) 241: BEGIN
                       OFFSET := 1;
                       REPEAT
                       ITEM_COUNT := 0;
                       OFFSET := OFFSET + 3;
                       REPEAT
                          ITEM_COUNT := ITEM_COUNT + 1;
                          FOR COUNT2 := 1 TO ORD(RESPONSE[OFFSET]) DO
```

```
                    WRITE( RESPONSE[OFFSET+COUNT2]);
                 COUNTER := ITEMS1L[ ITEM_COUNT ]-ORD(RESPONSE[OFFSET]);
                 IF COUNTER > 0 THEN
                    FOR COUNTER2 := 1 TO COUNTER DO
                       WRITE(' ');
                 WRITE('  ');
                 OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
              UNTIL ORD(RESPONSE[OFFSET]) = 255;
              WRITELN;
              OFFSET := OFFSET + 2;
              UNTIL ORD(RESPONSE[OFFSET]) = 246;
           END;

(* COMPLETION CODE RESPONSE *)

(* F6 *) 246: BEGIN
              WRITE('Completion Code is ');
              TOHEX(RESPONSE[6],HEXSTRING);
              FOR COUNT := 1 TO 2 DO WRITE(HEXSTRING[COUNT]);
              WRITE(' ');
              TOHEX(RESPONSE[5],HEXSTRING);
              FOR COUNT := 1 TO 2 DO WRITE(HEXSTRING[COUNT]);
              WRITELN;
              WRITELN('--> Refer to the iDBP reference manual.');
              WRITELN;WRITELN;
           END;

(* LIST RESPONSE *)

(* F8 *) 248: BEGIN
           IF ORD(RESPONSE[7]) <> 255 THEN
           BEGIN

             WRITELN;WRITELN;
             CASE ORD( RESPONSE[3]) OF

            144 : BEGIN
                     OFFSET := 7;
                     DBCOUNT := 0;
                     REPEAT

                     CASE ORD(RESPONSE[OFFSET+1]) OF

                      0: BEGIN
                            VIEWCOUNT := VIEWCOUNT + 1;
                            WRITE('          view # ',VIEWCOUNT:2,' ')
                         END;
                      1: BEGIN
                            FILECOUNT := 0;
                            DBCOUNT := DBCOUNT + 1;
                            WRITE('database # ',DBCOUNT:2,' ')
                         END;
                      3: BEGIN
                            VIEWCOUNT := 0;
                            FILECOUNT := FILECOUNT + 1;
                            WRITE('      file # ',FILECOUNT:2,' ' )
                         END;
                     END;
                     OFFSET := OFFSET + 9;
                     FOR COUNT := 1 TO ORD(RESPONSE[OFFSET]) DO
```

```
                        WRITE(RESPONSE[OFFSET+COUNT]);
                    WRITELN;
                    OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1

                    UNTIL ORD(RESPONSE[OFFSET])=255;
                    WRITELN
                  END;

              146 : BEGIN
                      WRITELN('List of Views :');WRITELN;
                      OFFSET := 7;
                      REPEAT
                         OFFSET := OFFSET + 9;
                         FOR COUNT := 1 TO ORD(RESPONSE[OFFSET]) DO
                             WRITE(RESPONSE[OFFSET+COUNT]);
                         WRITELN;
                         OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1
                      UNTIL ORD(RESPONSE[OFFSET])=255;
                      WRITELN
                    END;
              END
              END
              ELSE
                WRITELN('I have never heard of that database.')
            END;

(* DESCRIBE VIEW RESPONSE *)

(* F9 *) 249: BEGIN
              IF ORD( RESPONSE[7]) <> 255 THEN
              BEGIN
               DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
                  ITEMS1,ITEMS1L,VALS1,VALS2 );
               WRITELN;
               WRITE('View : ');
               FOR COUNT := 1 TO ITEM1L DO
                   WRITE( ITEM1[ COUNT ]);
               WRITELN;
               WRITE('Underlying Relation : ');
               FOR COUNT := 1 TO ITEM2L DO
                   WRITE( ITEM2[ COUNT ]);
               WRITELN;
               WRITELN('# of items = ',NUM_ITEMS:3);
               WRITELN;
               FOR COUNT := 1 TO NUM_ITEMS DO
               BEGIN
                  FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
                      WRITE( ITEMS1[ COUNT,COUNT2 ]);
                  SPACE_FILL := 20 - ITEMS1L[ COUNT ];
                  FOR COUNT2 := 1 TO SPACE_FILL DO
                      WRITE(' ');
                  CASE VALS1[ COUNT ] OF

                  0 : WRITE('Unsigned Integer   ');
                  1 : WRITE('Signed Integer     ');
                  2 : WRITE('Uninterpreted      ');
                  3 : WRITE('ASCII Alphanumeric ');
                  7 : WRITE('Record Pointer     ');
                  9 : WRITE('String Pointer     ');
                  64: WRITE('Zero Integer       ');
```

```
                        END;
                        WRITELN( VALS2[ COUNT ] )
                     END
                   END
                   ELSE
                      WRITELN('I have never heard of that view.')
                END;

      (* REMARK RESPONSE *)

      (* FC *) 252: BEGIN
                      WRITE('Echo : ');
                      FOR COUNTER := 1 TO ORD(RESPONSE[4])-1 DO
                          WRITE( RESPONSE[ COUNTER+6 ]);
                      WRITELN;WRITELN
                   END;

      OTHERWISE
               ;
      END
   END
END;


(*-----------------------------------------------*)

(* ATTACH *)

PROCEDURE ATTACH( ITEM:IDTYPE;ITEML:NUMTYPE;NUMBER_OF_ITEMS,
          OWN_USE,OTHERS_USE,WAIT_FLAG,ABORT_FLAG:INTEGER );

VAR   COUNT : INTEGER;


BEGIN
   INT1(0);
   INT1(1);
   BITSB;
   BITS( WAIT_FLAG*128 );
   BITS( ABORT_FLAG*64 );
   BITSE;
   FOR COUNT := 1 TO NUMBER_OF_ITEMS DO
   BEGIN
      ASC( ITEM[ COUNT ],ITEML[ COUNT ] );
      INT1(1);
      BITSB;
      BITS( OWN_USE*16 );
      BITS( OTHERS_USE );
      BITSE
   END;
   TERMINATE
END;


(* DEFINE DATABASE *)

PROCEDURE DEFINE_DATABASE( DBNAME:IDENT_STRING;
          DBNAMEL:INTEGER );
BEGIN
```

```
   INT1(96);
   ASC(DBNAME,DBNAMEL);
   TERMINATE
END;


(* DEFINE FILE *)

PROCEDURE DEFINE_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          PAGESIZE:INTEGER;VOLUMEID:IDENT_STRING;VOLUMEIDL:INTEGER;
          INITALLOC,MAXALLOC:INTEGER );

BEGIN
   INT1(64);
   ASC(FILENAME,FILENAMEL);
   INT1(1);
   INT1(PAGESIZE*128);
   ASC(VOLUMEID,VOLUMEIDL);
   INT1(2);
   INT2(INITALLOC);
   INT1(2);
   INT2(MAXALLOC);
   TERMINATE
END;


(* DEFINE SCHEMA *)

PROCEDURE DEFINE_SCHEMA( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          DSTYPE,REUSE,EXPAREA,SUBSTRLEN:INTEGER;
          VARITEM:INTEGER;VIEW:IDENT_STRING;VIEWL:INTEGER;
          NUM_ITEMS:INTEGER;
          DS_ID:IDTYPE;DS_IDL:NUMTYPE;
          FIXEDVAR:FIXEDVAR_TYPE;ITEMTYPE:ITEM_TYPE;
          ITEML:ITEML_TYPE );


VAR   COUNT: INTEGER;

BEGIN
   INT1(73);
   ASC(FILENAME,FILENAMEL);
   INT1(1);
   BITSB;BITS(DSTYPE*16);BITS(REUSE*8);BITSE;
   IF VIEWL <> 0 THEN
   (* A VIEW HAS BEEN SPECIFIED FOR THE SCHEMA *)
      ASC( VIEW,VIEWL )
   ELSE
   (* A SCHEMA HAS BEEN EXPLICITLY DEFINED *)
   BEGIN
      INT1(0);
      INT1(2);
      IF EXPAREA = 0 THEN
         INT2(SUBSTRLEN)
      ELSE
         INT2(EXPAREA);
      IF DSTYPE <> 0 THEN
         INT1(0)
      ELSE
      BEGIN
```

```
         INT1(2);
         INT2(VARITEM)
      END;
      FOR COUNT := 1 TO NUM_ITEMS DO
      BEGIN
         ASC(DS_ID[ COUNT ],DS_IDL[ COUNT ]);
         INT1(1);
         BITSB;BITS(FIXEDVAR[COUNT]*128);
         BITS( ITEMTYPE[ COUNT ] );BITSE;
         IF DSTYPE <> 0 THEN
            INT1(0)
         ELSE
         BEGIN
            INT1(1);
            INT1(ITEML[ COUNT ])
         END
      END
   END;
   TERMINATE
END;


(* DEFINE VIEW - CONNECT *)

PROCEDURE DEFINE_VIEW_CONNECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE1:IDENT_STRING;SOURCE1L:INTEGER;
          STRINGPTR:IDENT_STRING;STRINGPTRL:INTEGER;
          SOURCE2:IDENT_STRING;SOURCE2L:INTEGER );

BEGIN
   INT1(80);
   ASC( NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(32);
   ASC( SOURCE1,SOURCE1L );
   INT1(255);
   INT1(5);
   ASC( STRINGPTR,STRINGPTRL );
   ASC( SOURCE2,SOURCE2L );
   TERMINATE
END;

(* DEFINE VIEW - JOIN *)

PROCEDURE DEFINE_VIEW_JOIN( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
                            SOURCE1:IDENT_STRING;SOURCE1L:INTEGER;
                            ITEM1:IDENT_STRING;ITEM1L:INTEGER;
                            SOURCE2:IDENT_STRING;SOURCE2L:INTEGER;
                            ITEM2:IDENT_STRING;ITEM2L:INTEGER );

BEGIN
   INT1(80);
   ASC( NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(16);
   ASC( SOURCE1,SOURCE1L );
   INT1(255);
   INT1(5);
   ASC( ITEM1,ITEM1L );
   ASC( SOURCE2,SOURCE2L );
```

```
   INT1(255);
   INT1(5);
   ASC( ITEM2,ITEM2L );
   TERMINATE
END;


(* DEFINE VIEW - ORDER *)

PROCEDURE DEFINE_VIEW_ORDER( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          ORDER_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE;
          ASC_OR_DESC:INTEGER );


VAR   COUNT: INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(64);
   ASC(SOURCE,SOURCEL);
   FOR COUNT := 1 TO ORDER_NUM DO
   BEGIN
      INT1(255);
      INT1(5);
      ASC(ITEMS1[ COUNT ],ITEMS1L[ COUNT ]);
      INT1(1);
      INT1(ASC_OR_DESC*128)
   END;
   TERMINATE
END;


(* DEFINE VIEW - PROJECT *)

PROCEDURE DEFINE_VIEW_PROJECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          INC_EXC:INTEGER;SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          ALL_INDICATOR:BOOLEAN;PROJECT_NUM:INTEGER;
          ITEM:IDTYPE;ITEML:NUMTYPE );


VAR   COUNT:  INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL);
   INT1(1);
   IF INC_EXC = 0 THEN
     INT1(96)
   ELSE
     INT1(112);
   ASC(SOURCE,SOURCEL);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
```

```
   BEGIN
      FOR COUNT := 1 TO PROJECT_NUM DO
      BEGIN
         INT1(255);
         INT1(5);
         ASC(ITEM[ COUNT ],ITEML[ COUNT ])
      END
   END;
   TERMINATE
END;

(* DEFINE VIEW - SELECT *)

PROCEDURE DEFINE_VIEW_SELECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          COMPARATOR:NUMTYPE;REC_SEARCH:INTEGER;
          COMPARE_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE;
          ITEMS2:IDTYPE;ITEMS2L:NUMTYPE );

VAR   COUNT:  INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL);
   INT1(1);
   INT1(48);
   ASC(SOURCE,SOURCEL);

   (* CREATE A BLOCK FOR EACH COMPARATOR W/OPERANDS *)

   FOR COUNT := 1 TO COMPARE_NUM DO
   BEGIN
      IF COMPARATOR[ COUNT ] IN [ 1,11 ] THEN
      BEGIN
         INT1(2);
         INT1( COMPARATOR[ COUNT ] );
         INT1( REC_SEARCH*128);
         INT1(255);
         INT1(5);
         ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ])
       END

      ELSE

      BEGIN
         INT1(2);
         INT1( COMPARATOR[ COUNT ] );
         INT1( REC_SEARCH*128 );
         INT1(255);
         INT1(5);
         ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ]);
         ASC( ITEMS2[ COUNT ],ITEMS2L[ COUNT ])
      END;
      IF COUNT < COMPARE_NUM THEN
      (* INSERT AN 'AND' OPERATION *)

      BEGIN
         INT1(2);
         INT1(250);
         INT1(0)
```

```
      END
   END;

   TERMINATE
END;

(* DELETE DATABASE *)

PROCEDURE DELETE_DATABASE( DBNAME:IDENT_STRING;DBNAMEL:INTEGER );

BEGIN
   INT1(97);
   ASC(DBNAME,DBNAMEL);
   TERMINATE
END;

(* DELETE FILE *)

PROCEDURE DELETE_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER );

BEGIN
   INT1(66);
   ASC(FILENAME,FILENAMEL);
   TERMINATE
END;


(* DELETE VIEW *)

PROCEDURE DELETE_VIEW( VIEW:IDENT_STRING;VIEWL:INTEGER );

BEGIN
   INT1(82);
   ASC( VIEW,VIEWL );
   TERMINATE
END;


(* DESCRIBE VIEW *)

PROCEDURE DESCRIBE_VIEW( VIEW:IDENT_STRING;VIEWL:INTEGER;
          ALL_INDICATOR:BOOLEAN );

BEGIN
   INT1(154);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( VIEW,VIEWL );
   TERMINATE
END;


(* DETACH *)

PROCEDURE DETACH( VIEW:IDENT_STRING;VIEWL:INTEGER;ALL_INDICATOR:BOOLEAN );
```

```
BEGIN
   INT1(1);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( VIEW,VIEWL );
   TERMINATE
END;

(* END CURSOR *)

PROCEDURE END_CURSOR( CURSOR_NUM:INTEGER;ALL_INDICATOR:BOOLEAN );
BEGIN
   INT1(3);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
   BEGIN
      INT1(1);
      INT1(CURSOR_NUM)
   END;
   TERMINATE
END;


(* FETCH *)

PROCEDURE FETCH( CURSOR_NUM:INTEGER;COUNT:INTEGER );

BEGIN
   INT1(16);
   INT1(1);
   INT1(CURSOR_NUM);
   IF COUNT = 0 THEN

   (* FETCH ALL *)

   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE

   (* FETCH A SPECIFIC NUMBER OF RECORDS *)

   BEGIN
      INT1(2);
      INT2(COUNT)
   END;
   TERMINATE
END;


(* FREE *)
```

```
PROCEDURE FREE( VIEW:IDENT_STRING;VIEWL:INTEGER;ALL_INDICATOR:BOOLEAN );

BEGIN
   INT1(1);
   IF ALL_INDICATOR THEN
   (* FREE ALL VIEWS *)
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
   (* FREE A SPECIFIC VIEW *)
      ASC( VIEW,VIEWL );
   TERMINATE
END;


(* KEEP DATABASE *)

PROCEDURE KEEP_DATABASE( OLDDB:IDENT_STRING;OLDDBL:INTEGER;
          NEWDB:IDENT_STRING;NEWDBL:INTEGER );

BEGIN
   INT1(100);
   ASC( OLDDB,OLDDBL );
   ASC( NEWDB,NEWDBL );
   TERMINATE
END;


(* KEEP FILE *)

PROCEDURE KEEP_FILE( OLDFILE:IDENT_STRING;OLDFILEL:INTEGER;
          NEWFILE:IDENT_STRING;NEWFILEL:INTEGER;
          DATABASE:IDENT_STRING;DATABASEL:INTEGER );

BEGIN
   INT1(65);
   ASC( OLDFILE,OLDFILEL );
   ASC( NEWFILE,NEWFILEL );
   IF DATABASEL = 0 THEN
      INT1(0)
   ELSE
      ASC( DATABASE,DATABASEL );
   TERMINATE
END;


(* KEEP VIEW *)

PROCEDURE KEEP_VIEW( OLDVIEW:IDENT_STRING;OLDVIEWL:INTEGER;
          NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER );

BEGIN
   INT1(81);
   ASC( OLDVIEW,OLDVIEWL );
   ASC( NEWVIEW,NEWVIEWL );
   TERMINATE
END;
```

```
(* LIST DATABASE *)

PROCEDURE LIST_DATABASE( DBNAME:IDENT_STRING;DBNAMEL:INTEGER;
          ALL_ENTITIES,ALL_DATABASES: BOOLEAN );

BEGIN
   INT1(144);
   IF ALL_DATABASES THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( DBNAME,DBNAMEL );
   INT1(1);
   IF NOT ALL_ENTITIES THEN
      INT1(0)
   ELSE
      INT1(160);
   TERMINATE
END;


(* LIST FILE *)

PROCEDURE LIST_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          ALL_INDICATOR:BOOLEAN );
BEGIN
   INT1(145);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( FILENAME,FILENAMEL );
   INT1(1);
   INT1(255);
   TERMINATE
END;


(* LIST VIEWS *)

PROCEDURE LIST_VIEWS;

BEGIN
   INT1(146);
   INT1(255);
   INT1(6);
   TERMINATE
END;

(* REMARK *)

PROCEDURE REMARK( A_WORD:IDENT_STRING;A_WORDL:INTEGER;DESTINATION:INTEGER );

BEGIN
```

```
   INT1(58);
   INT1(1);
   INT1(DESTINATION);
   ASC(A_WORD,A_WORDL);
   TERMINATE
END;


(* SUBMIT KEYS *)

PROCEDURE SUBMIT_KEYS( KEY:IDENT_STRING;KEYL:INTEGER );

BEGIN
   INT1(7);
   ASC( KEY,KEYL );
   TERMINATE
END;

(* START CURSOR *)

PROCEDURE START_CURSOR( CURSOR_NUM:INTEGER; VIEW:IDENT_STRING;
          VIEWL:INTEGER;MODE:INTEGER;DIRECTION:INTEGER;
          RETEST:INTEGER );

BEGIN
   INT1(2);
   INT1(1);
   INT1(CURSOR_NUM);
   ASC(VIEW,VIEWL);
   INT1(1);
   BITSB;
   BITS(MODE*128);
   BITS(DIRECTION*64);
   BITS(RETEST*32);
   BITSE;
   TERMINATE
END;

PROCEDURE STORE( CURSOR_NUM:INTEGER;INTEGRITY:BOOLEAN;
                 ITEM_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE );

VAR   COUNT:INTEGER;

BEGIN
   INT1(18);
   INT1(1);
   INT1(CURSOR_NUM);
   INT1(1);
   IF INTEGRITY THEN
      INT1(0)
   ELSE
      INT1(128);
   FOR COUNT := 1 TO ITEM_NUM DO
   BEGIN
      ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ])
   END;
   TERMINATE
END;

$
```

# APPENDIX B - DBPQL Grammar File

```
$ type [intel.dbpql]dbpql.grm
<GOAL> ::= <QUERY> <EOLN>
 ;
*=================================
* QUERY TYPES
*=================================
<QUERY> ::=
 ;
<QUERY> ::= <ATTACH>
 ;
<QUERY> ::= <CREATE_DATABASE>
 ;
<QUERY> ::= <CREATE_RELATION>
 ;
<QUERY> ::= <CREATE_VIEWC>
 ;
<QUERY> ::= <CREATE_VIEWJ>
 ;
<QUERY> ::= <CREATE_VIEWO>
 ;
<QUERY> ::= <CREATE_VIEWP>
 ;
<QUERY> ::= <CREATE_VIEWS>
 ;
<QUERY> ::= <DELETE_DATABASE>
 ;
<QUERY> ::= <DELETE_RELATION>
 ;
<QUERY> ::= <DELETE_VIEW>
 ;
<QUERY> ::= <DETACH>
 ;
<QUERY> ::= <DISPLAY>
 ;
<QUERY> ::= <ECHO>
 ;
<QUERY> ::= <HELP>
 ;
<QUERY> ::= <INPUT>
 ;
<QUERY> ::= <LIST_DBS>
 ;
<QUERY> ::= <LIST_DB>
 ;
<QUERY> ::= <LIST_VIEWS>
 ;
<QUERY> ::= <LIST_VIEW>
 ;
<QUERY> ::= <LOAD>
 ;
<QUERY> ::= <PERFORM_COMMAND>
 ;
<QUERY> ::= <TRACE_COMMAND>
 ;
*=================================
* ATTACH
*=================================
<ATTACH> ::= <ATTACHK> <VIEWS> <PERMISSION>
 BEGIN
```

```
    START;
    FREE( BLANK_IDENT,1,TRUE );
    ATTACH( ITEMS3,ITEMS3L,IDCOUNT,PERMISSION,0,0,0 );
    QUERY
 END;
<ATTACHK> ::= ATTACH
 IDCOUNT := 0;
<VIEWS> ::= <ATTACH_VIEW>
 ;
<VIEWS> ::= <VIEWS> <ATTACH_VIEW>
 ;
<ATTACH_VIEW> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS3[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS3L[ IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<PERMISSION> ::= READ
 PERMISSION := 1;
<PERMISSION> ::= WRITE
 PERMISSION := 2;
<PERMISSION> ::= RW
 PERMISSION := 3;
<PERMISSION> ::= ADMIN
 PERMISSION := 4;
*==================================
* CREATE DATABASE
*==================================
<CREATE_DATABASE> ::= <CREATE_DATABASEK> <CD_DBNAME>
 BEGIN
    START;
    DEFINE_DATABASE(DBNAME,DBNAMEL);
    KEEP_DATABASE(DBNAME,DBNAMEL,DBNAME,DBNAMEL);
    QUERY
 END;
<CREATE_DATABASEK> ::= CREATE DATABASE
 ;
<CD_DBNAME> ::= <ID>
 BEGIN
    DBNAME := SSTACK[SP].IDNAME;
    DBNAMEL:= SSTACK[SP].IDLEN
 END;
*==================================
* CREATE RELATION
*==================================
<CREATE_RELATION> ::= <CREATE_RELATIONK> <DF_REST>
 BEGIN
    START;
    DEFINE_FILE( DF_FILENAME,DF_FILENAMEL,DF_PAGESIZE,
                 DF_VOLUMEID,DF_VOLUMEIDL,
                 DF_INITALLOC,DF_MAXALLOC );

    DEFINE_SCHEMA( DF_FILENAME,DF_FILENAMEL,DS_TYPE,DS_REUSE,
                   DS_EXPAREA,DS_SUBSTRLEN,DS_VARITEM,DS_VIEW,DS_VIEWL,
                   IDCOUNT,DS_ID,DS_IDL,DS_FIXEDVAR,
                   DS_ITEMTYPE,DS_ITEML );

    KEEP_FILE( DF_FILENAME,DF_FILENAMEL,DF_FILENAME,DF_FILENAMEL,
               DBNAME,DBNAMEL );
    WRITELN('Ok, View ',DF_FILENAME:DF_FILENAMEL,' has been created.');
```

```
    QUERY
 END;
<CREATE_RELATIONK> ::= CREATE RELATION
 BEGIN

    (* DEFAULTS *)

    (* DEFINE FILE DEFAULTS *)

    DF_FILENAME := 'FILE1       ';
    DF_FILENAMEL := 5;
    DF_PAGESIZE := 0;
    DF_VOLUMEID := 'DBPSYS      ';
    DF_VOLUMEIDL := 6;
    DF_INITALLOC := 20;
    DF_MAXALLOC := 0;

    (* DEFINE SCHEMA DEFAULTS *)

    DS_TYPE := 0;
    DS_REUSE := 0;
    DS_VIEWL := 0;
    DS_EXPAREA := 20;
    DS_SUBSTRLEN := 80;
    DS_VARITEM := 20;
    IDCOUNT := 0;

 END;
<DF_REST> ::= <ID> IN <ID> <DF_REST2>
 BEGIN
    DF_FILENAME := SSTACK[MP].IDNAME;
    DF_FILENAMEL:= SSTACK[MP].IDLEN;
    DBNAME := SSTACK[MP+2].IDNAME;
    DBNAMEL:= SSTACK[MP+2].IDLEN
 END;
<DF_REST2> ::= USING SCHEMA <SCHEMA> <OPTIONS>
 ;
<DF_REST2> ::= USING VIEW <ID>
 BEGIN
    DS_VIEW := SSTACK[SP].IDNAME;
    DS_VIEWL:= SSTACK[SP].IDLEN;
 END;
<SCHEMA> ::= <ITEM>
 ;
<SCHEMA> ::= <SCHEMA> <ITEM>
 ;
<ITEM> ::= <ITEM_ID> <DATA_TYPE> <FIXED_VAR> <ITEM_LENGTH>
 ;
<ITEM_ID> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    DS_ID[ IDCOUNT ] := SSTACK[MP].IDNAME;
    DS_IDL[ IDCOUNT ] := SSTACK[MP].IDLEN
 END;
<FIXED_VAR> ::= FIXED
 DS_FIXEDVAR[ IDCOUNT ] := 0;
<FIXED_VAR> ::= VAR
 DS_FIXEDVAR[ IDCOUNT ] := 1;
<ITEM_LENGTH> ::= <NO>
```

```
 DS_ITEML[ IDCOUNT ] := SSTACK[SP].IVAL;
<DATA_TYPE> ::= UNSIGNED_INT
 DS_ITEMTYPE[ IDCOUNT ] := 0;
<DATA_TYPE> ::= SIGNED_INT
 DS_ITEMTYPE[ IDCOUNT ] := 1;
<DATA_TYPE> ::= INTEGER
 DS_ITEMTYPE[ IDCOUNT ] := 1;
<DATA_TYPE> ::= UNINTERPRET
 DS_ITEMTYPE[ IDCOUNT ] := 2;
<DATA_TYPE> ::= ASCII
 DS_ITEMTYPE[ IDCOUNT ] := 3;
<DATA_TYPE> ::= RECORD_PTR
 DS_ITEMTYPE[ IDCOUNT ] := 7;
<DATA_TYPE> ::= STRING_PTR
 DS_ITEMTYPE[ IDCOUNT ] := 9;
<OPTIONS> ::=
 ;
<OPTIONS> ::= <OPTIONS> <OPTION>
 ;
<OPTION> ::= SMALLPAGE
 DF_PAGESIZE := 0;
<OPTION> ::= LARGEPAGE
 DF_PAGESIZE := 1;
<OPTION> ::= VOLUME <ID>
 BEGIN
    DF_VOLUMEID := SSTACK[SP].IDNAME;
    DF_VOLUMEIDL:= SSTACK[SP].IDLEN;
 END;
<OPTION> ::= INITALLOC <NO>
 DF_INITALLOC := SSTACK[SP].IVAL;
<OPTION> ::= EXPANDFILE
 DF_MAXALLOC := 0;
<OPTION> ::= MAXALLOC <NO>
 DF_MAXALLOC := SSTACK[SP].IVAL;
<OPTION> ::= STRUCTURED
 DS_TYPE := 0;
<OPTION> ::= UN_CLEAR
 DS_TYPE := 1;
<OPTION> ::= UN_COMPLEX
 DS_TYPE := 2;
<OPTION> ::= UN_UNINTERP
 DS_TYPE := 3;
<OPTION> ::= UN_BACKUP
 DS_TYPE := 4;
<OPTION> ::= UN_ROLLF
 DS_TYPE := 5;
<OPTION> ::= RE_USE
 DS_REUSE := 0;
<OPTION> ::= NORE_USE
 DS_REUSE := 1;
<OPTION> ::= EXP_AREA <NO>
 DS_EXPAREA := SSTACK[SP].IVAL;
<OPTION> ::= SUBSTR_LEN <NO>
 DS_SUBSTRLEN := SSTACK[SP].IVAL;
<OPTION> ::= VARITEM <NO>
 DS_VARITEM := SSTACK[SP].IVAL;
*================================
* CREATE VIEW - CONNECT
*================================
<CREATE_VIEWC> ::= <CREATE_VIEWCK> <CVC_REST>
```

```
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEWS *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ITEMS3[2] := SOURCE2;
    ITEMS3L[2] := SOURCE2L;
    ATTACH( ITEMS3,ITEMS3L,2,3,0,0,0 );
    DEFINE_VIEW_CONNECT( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                STRPTR,STRPTRL,SOURCE2,SOURCE2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWCK> ::= CREATE CONNECT VIEW
 ;
<CVC_REST> ::= <NEWVIEW> FROM <SOURCE1> <STRPTR> <SOURCE2>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN;
    STRPTR := SSTACK[MP+3].IDNAME;
    STRPTRL:= SSTACK[MP+3].IDLEN;
    SOURCE2 := SSTACK[SP].IDNAME;
    SOURCE2L := SSTACK[SP].IDLEN
 END;
<SOURCE1> ::= <ID>
 ;
<SOURCE2> ::= <ID>
 ;
<NEWVIEW> ::= <ID>
 ;
<STRPTR> ::= <ID>
 ;
*=================================
* CREATE VIEW - JOIN
*=================================
<CREATE_VIEWJ> ::= <CREATE_VIEWJK> <CVJ_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEWS *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ITEMS3[2] := SOURCE2;
    ITEMS3L[2] := SOURCE2L;
    ATTACH( ITEMS3,ITEMS3L,2,3,0,0,0 );
    DEFINE_VIEW_JOIN( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                ITEM1,ITEM1L,SOURCE2,SOURCE2L,
                ITEM2,ITEM2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWJK> ::= CREATE JOIN VIEW
 ;
<CVJ_REST> ::= <NEWVIEW> FROM <SOURCE1> <ID> <SOURCE2> <ID>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
```

```
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN;
    ITEM1 := SSTACK[MP+3].IDNAME;
    ITEM1L:= SSTACK[MP+3].IDLEN;
    SOURCE2 := SSTACK[MP+4].IDNAME;
    SOURCE2L := SSTACK[MP+4].IDLEN;
    ITEM2 := SSTACK[SP].IDNAME;
    ITEM2L:= SSTACK[SP].IDLEN
 END;
*==================================
* CREATE VIEW - ORDER
*==================================
<CREATE_VIEWO> ::= <CREATE_VIEWOK> <CVO_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_ORDER( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                IDCOUNT,ITEMS1,ITEMS1L,ASC_OR_DESC );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWOK> ::= CREATE ORDER VIEW
 BEGIN
    IDCOUNT := 0;
    ASC_OR_DESC := 0
 END;
<CVO_REST> ::= <NEWVIEW> FROM <SOURCE1> <ORDER_ITEMS> <ASC_OR_DESC>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<ORDER_ITEMS> ::= <ORDER_ITEM>
 ;
<ORDER_ITEMS> ::= <ORDER_ITEMS> <ORDER_ITEM>
 ;
<ORDER_ITEM> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<ASC_OR_DESC> ::=
 ;
<ASC_OR_DESC> ::= ASCENDING
 ASC_OR_DESC := 0;
<ASC_OR_DESC> ::= DESCENDING
 ASC_OR_DESC := 1;
*==================================
* CREATE VIEW - PROJECT
*==================================
<CREATE_VIEWP> ::= <CREATE_VIEWPK> <CVP_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
```

```
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_PROJECT( NEWVIEW,NEWVIEWL,INC_EXC,SOURCE1,SOURCE1L,
                         ALL_INDICATOR,IDCOUNT,ITEMS1,ITEMS1L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWPK> ::= CREATE PROJECT VIEW
 BEGIN
    IDCOUNT := 0;
    ALL_INDICATOR := FALSE;
    INC_EXC := 0
 END;
<CVP_REST> ::= <NEWVIEW> FROM <SOURCE1> <INC_EXC> <PROJECT_IDS>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<PROJECT_IDS> ::= ALL-ITEMS
 ALL_INDICATOR := TRUE;
<PROJECT_IDS> ::= <PROJECT_ID>
 ;
<PROJECT_IDS> ::= <PROJECT_IDS> <PROJECT_ID>
 ;
<PROJECT_ID> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<INC_EXC> ::=
 ;
<INC_EXC> ::= INCLUDING
 INC_EXC := 0;
<INC_EXC> ::= EXCLUDING
 INC_EXC := 1;
*==================================
* CREATE VIEW - SELECT
*==================================
<CREATE_VIEWS> ::= <CREATE_VIEWSK> <CVS_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_SELECT( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,COMPARATOR,
                REC_SEARCH,IDCOUNT,ITEMS1,ITEMS1L,ITEMS2,ITEMS2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWSK> ::= CREATE SELECT VIEW
 BEGIN
    REC_SEARCH := 0;
    IDCOUNT := 0
```

```
 END;
<CVS_REST> ::= <NEWVIEW> FROM <SOURCE1> WHERE <WHERE_CLAUSE> <SELECT_OPTIONS>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<WHERE_CLAUSE> ::= <SINGLE_CLAUSE>
 ;
<WHERE_CLAUSE> ::= <WHERE_CLAUSE> AND <SINGLE_CLAUSE>
 ;
<SINGLE_CLAUSE> ::= <FIRST> <BINARY> <SECOND>
 ;
<FIRST> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[ SP ].IDNAME;
    ITEMS1L[ IDCOUNT ] := SSTACK[ SP ].IDLEN
 END;
<SECOND> ::= <ID>
 BEGIN
    ITEMS2[ IDCOUNT ] := SSTACK[ SP ].IDNAME;
    ITEMS2L[ IDCOUNT ] := SSTACK[ SP ].IDLEN
 END;
<SECOND> ::= <NO>
 BEGIN
    NUM_TO_ASCII( SSTACK[SP].IVAL,ITEM1,ITEM1L );
    ITEMS2[ IDCOUNT ] := ITEM1 ;
    ITEMS2L[ IDCOUNT ] := ITEM1L
 END;
<SINGLE_CLAUSE> ::= <FIRST> <UNARY>
 ;
<BINARY> ::= =
 COMPARATOR[ IDCOUNT ] := 20;
<BINARY> ::= <>
 COMPARATOR[ IDCOUNT ] := 30;
<BINARY> ::= <
 COMPARATOR[ IDCOUNT ] := 40;
<BINARY> ::= <=
 COMPARATOR[ IDCOUNT ] := 60;
<BINARY> ::= >=
 COMPARATOR[ IDCOUNT ] := 50;
<BINARY> ::= >
 COMPARATOR[ IDCOUNT ] := 70;
<UNARY> ::= IS VALUED
 COMPARATOR[ IDCOUNT ] := 1;
<UNARY> ::= IS NULL
 COMPARATOR[ IDCOUNT ] := 11;
<SELECT_OPTIONS> ::= PERFORM
 REC_SEARCH := 0;
<SELECT_OPTIONS> ::= NOPERFORM
 REC_SEARCH := 1;
<SELECT_OPTIONS> ::=
 ;
*==================================
* DELETE DATABASE
*==================================
<DELETE_DATABASE> ::= <DELETE_DATABASEK> <ID>
 BEGIN
```

```
    START;
    DELETE_DATABASE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_DATABASEK> ::= DELETE DATABASE
 ;
*================================
* DELETE RELATION
*================================
<DELETE_RELATION> ::= <DELETE_RELATIONK> <ID>
 BEGIN
    START;
    (* ATTACH THE IDENTITY VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SSTACK[SP].IDNAME;
    ITEMS3L[1] := SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,4,0,0,0 );
    DELETE_FILE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_RELATIONK> ::= DELETE RELATION
 ;
*================================
* DELETE VIEW
*================================
<DELETE_VIEW> ::= <DELETE_VIEWK> <ID>
 BEGIN
    (* FIRST FIND THE UNDERLYING FILE- IDENTITY VIEW NAME *)
    (* THIS IDENTITY VIEW MUST BE ATTACHED WITH 'ADMIN ' *)

    START;
    DESCRIBE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,FALSE );
    SEND;
    RECV( RESPONSE,TOTAL_BYTES,MORE );
    DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
                 ITEMS1,ITEMS1L,VALS1,VALS2 );
    START;
    FREE( BLANK_IDENT,1,TRUE );
    ITEMS3[1] := ITEM2;
    ITEMS3L[1] := ITEM2L;
    ATTACH( ITEMS3,ITEMS3L,1,4,0,0,0 );
    DELETE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_VIEWK> ::= DELETE VIEW
 ;
*================================
* DETACH
*================================
<DETACH> ::= <DETACHK> <DETACH_WHAT>
 BEGIN
    START;
    DETACH( ITEM1,ITEM1L,ALL_INDICATOR );
    QUERY
 END;
<DETACHK> ::= DETACH
 ALL_INDICATOR := FALSE;
<DETACH_WHAT> ::= <ID>
 BEGIN
    ITEM1 := SSTACK[SP].IDNAME;
```

```
    ITEM1L:= SSTACK[SP].IDLEN
 END;
<DETACH_WHAT> ::= ALL
 ALL_INDICATOR := TRUE;
*==================================
* DISPLAY
*==================================
<DISPLAY> ::= DISPLAY <ID>
 BEGIN
    START;
    DESCRIBE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,FALSE );
    SEND;
    RECV( RESPONSE,TOTAL_BYTES,MORE );
    DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
                 ITEMS1,ITEMS1L,VALS1,VALS2 );
    (* PROVIDE A HEADER FOR THE VIEW DISPLAY *)
    WRITELN;
    WRITELN;
    FOR COUNT := 1 TO NUM_ITEMS DO
    BEGIN
       FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
           WRITE( ITEMS1[ COUNT,COUNT2 ] );
       COUNTER := VALS2[ COUNT ] - ITEMS1L[ COUNT ];
       IF COUNTER > 0 THEN
       BEGIN
          FOR COUNTER2 := 1 TO COUNTER DO
              WRITE(' ')
       END;
       WRITE('  ')
    END;
    WRITELN;
    (* DRAW THE UNDERLINING *)
    FOR COUNT := 1 TO NUM_ITEMS DO
    BEGIN
       FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
           WRITE('_');
       COUNTER := VALS2[ COUNT ] - ITEMS1L[ COUNT ];
       IF COUNTER > 0 THEN
       BEGIN
          FOR COUNTER2 := 1 TO COUNTER DO
              WRITE(' ')
       END;
       WRITE('  ')
    END;
    WRITELN;
    WRITELN;

    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SSTACK[SP].IDNAME;
    ITEMS3L[1]:= SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    START_CURSOR(1,SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,
          0,0,1 );
    FETCH(1,0);
    END_CURSOR(1,TRUE);
    QUERY
 END;
*==================================
```

```
* ECHO
*==================================
<ECHO> ::= <ECHOK> <ID>
 BEGIN
    START;
    REMARK( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,1);
    QUERY
 END;
<ECHOK> ::= ECHO
 ;
*==================================
* HELP
*==================================
<HELP> ::= <HELPK> <HELP_COMMAND>
 ;
<HELPK> ::= HELP
 ;
<HELP_COMMAND> ::=
 BEGIN
   WRITELN;WRITELN;
   WRITELN('The following commands are available :');
   WRITELN;
   WRITELN('CREATE   DELETE   DISPLAY   ECHO  INPUT   LIST');
   WRITELN('LOAD     PERFON   PERFOFF   TRACEON');
   WRITELN('TRACEOFF');
   WRITELN;
   WRITELN('General Notes :');WRITELN;
   WRITELN('1. To continue lines of input use a dash( - ) at the end');
   WRITELN('   of each line to be continued.');
   WRITELN('2. To exit DBPQL, just enter X at the prompt.');
   WRITELN;
   WRITELN('For a specific help, type HELP <command>')
 END;
<HELP_COMMAND> ::= CREATE
 BEGIN
    WRITELN;WRITELN;
    WRITELN('You can create several different entities :');WRITELN;
    WRITELN('CREATE DATABASE');
    WRITELN('CREATE RELATION');
    WRITELN('CREATE CONNECT VIEW');
    WRITELN('CREATE JOIN VIEW');
    WRITELN('CREATE PROJECT VIEW');
    WRITELN('CREATE SELECT VIEW');
    WRITELN('CREATE ORDER VIEW');WRITELN;
    WRITELN('To get more help on any one of these,');
    WRITELN('type HELP <one of the above lines>')
 END;
<HELP_COMMAND> ::= CREATE DATABASE
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE DATABASE <DBNAME>');
    WRITELN('where <DBNAME> is the name of the database to be created.')
 END;
<HELP_COMMAND> ::= CREATE RELATION
 BEGIN
    WRITELN;WRITELN;
    WRITELN('You can define a new schema for a relation as follows:');
    WRITELN('CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA');
    WRITELN('  { <ITEMNAME> <TYPE1> <TYPE2> <LENGTH> }');
    WRITELN('  where : <TYPE1> = SIGNED_INT or INTEGER');
```

```
    WRITELN('                    UNSIGNED_INT');
    WRITELN('                    UNINTERPRET');
    WRITELN('                    ASCII');
    WRITELN('                    RECORD_PTR');
    WRITELN('                    STRING_PTR');
    WRITELN('          <TYPE2> = FIXED');
    WRITELN('                    VAR');
    WRITELN('          <LENGTH>= of item in bytes');
    WRITELN;
    WRITELN('You can also create a new relation which uses the');
    WRITELN('already defined schema of any given view :');
    WRITELN;
    WRITELN('CREATE RELATION <RELNAME> USING VIEW <OLD_VIEW>')
 END;
<HELP_COMMAND> ::= CREATE CONNECT VIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE CONNECT VIEW <NEW_VIEW> <REST>');
    WRITELN('<REST>      = FROM <SOURCE_VIEW1> <STRING_PTR> <SOURCE_VIEW2>')
 END;
<HELP_COMMAND> ::= CREATE JOIN VIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE JOIN VIEW <NEW_VIEW> <REST>');
    WRITELN('<REST>      = FROM <SOURCE_VIEW1> <ITEM1> <SOURCE_VIEW2> <ITEM2>')
 END;
<HELP_COMMAND> ::= CREATE PROJECT VIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE PROJECT VIEW <NEW_VIEW> <REST>');
    WRITELN('<REST>      = FROM <SOURCE_VIEW> <INC_EXC> <PROJECT_ITEMS>');
    WRITELN('where <INC_EXC>= INCLUDING( default ) or EXCLUDING');
    WRITELN('and <PROJECT_ITEMS> = sequence of items to project')
 END;
<HELP_COMMAND> ::= CREATE SELECT VIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE SELECT VIEW <NEW_VIEW> <REST>');
    WRITELN('<REST>      = FROM <SOURCE_VIEW> WHERE <WHERE_CLAUSE> <OPTIONS>');
    WRITELN;
    WRITELN('<WHERE_CLAUSE>  = sequence of <BINARY> or <UNARY> clause(s),');
    WRITELN('separated by AND');WRITELN;
    WRITELN('where <BINARY>  = <ITEM> <BINARY_OP> <VALUE>');
    WRITELN('<BINARY_OP> =    = , <> , < , > , <= , >= ');WRITELN;
    WRITELN('and   <UNARY>   = <ITEM> <UNARY_OP>');
    WRITELN('<UNARY_OP>  =    IS VALUED or IS NULL')
 END;
<HELP_COMMAND> ::= CREATE ORDER VIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE ORDER VIEW <NEW_VIEW> <REST>');
    WRITELN('where <REST>   = FROM <SOURCE_VIEW> <ORDER_ITEMS> <ASC_OR_DESC>');
    WRITELN;
    WRITELN('<ORDER_ITEMS>  = sequence of items to sort');
    WRITELN('<ASC_OR_DESC>  = ASCENDING( default ) or DESCENDING')
 END;
<HELP_COMMAND> ::= DELETE
 BEGIN
    WRITELN;WRITELN;
    WRITELN('You may delete a DATABASE, RELATION, or VIEW by');
```

```
    WRITELN('saying DELETE <which-type> <identifier of thing to delete>')
 END;
<HELP_COMMAND> ::= DISPLAY
 BEGIN
    WRITELN;WRITELN;
    WRITELN('This command allows you to see a view on your terminal.');
    WRITELN('Just say, DISPLAY <view name>')
 END;
<HELP_COMMAND> ::= ECHO
 BEGIN
    WRITELN;WRITELN;
    WRITELN('This command serves as a simple test to see if the DBP');
    WRITELN('is up and running. Say ECHO <any-word> and you should');
    WRITELN('receive an echo of the word you received. If you do not');
    WRITELN('then the communications from the VAX to the DBP has not');
    WRITELN('been initialized correctly.')
 END;
<HELP_COMMAND> ::= INPUT
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Take all further command input from the file referenced');
    WRITELN('using logical name DBPIN. Assign the logical name prior');
    WRITELN('to invoking DBPQL. As an example :');WRITELN;
    WRITELN('$ ASSIGN DBP.DAT DBPIN');
    WRITELN('will assign the file DBP.DAT to the logical name DBPIN.');
 END;
<HELP_COMMAND> ::= LIST
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Type help <one-of-the-following> for further help on list:');
    WRITELN('LISTVIEW');
    WRITELN('LISTVIEWS');
    WRITELN('LISTDB');
    WRITELN('LISTDBS');
 END;
<HELP_COMMAND> ::= LISTVIEWS
 BEGIN
    WRITELN;WRITELN;
    WRITELN('LISTVIEWS gives a list of all available views.')
 END;
<HELP_COMMAND> ::= LISTDB
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Two forms: LISTDB <DBNAME>  and  LISTDB ALL' );
    WRITELN('The database entities : file and view are listed for');
    WRITELN('the given database(s).');WRITELN;
    WRITELN('See HELP LISTDBS for a brief form which lists only database names.')
 END;
<HELP_COMMAND> ::= LISTDBS
 BEGIN
    WRITELN;WRITELN;
    WRITELN('LISTDBS lists all currently available iDBP databases.');
    WRITELN('For a list containing file and view names, see HELP LISTDB.')
 END;
<HELP_COMMAND> ::= LISTVIEW
 BEGIN
    WRITELN;WRITELN;
    WRITELN('LISTVIEW <view name> gives the structure of the specified view.');
    WRITELN('The item names, data types, and item lengths are printed.')
 END;
```

```
<HELP_COMMAND> ::= LOAD
 BEGIN
    WRITELN;WRITELN;
    WRITELN('LOAD <view name> <tuples>');WRITELN;
    WRITELN('where <tuples> is a sequence of tuples and each tuple');
    WRITELN('is in the form :  [ valuel value2 value3 ... ]');
    WRITELN;WRITELN('Note the brackets must be included.')
 END;
<HELP_COMMAND> ::= PERFON
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns on the Performance Tracing.');
    WRITELN('The following statistics are measured :');WRITELN;
    WRITELN('1. Elapsed Clock Time');WRITELN('2. Elapsed CPU Time');
    WRITELN('3. Buffered I/O Count');WRITELN('4. Direct I/O Count');
    WRITELN('5. Page Fault Count')
 END;
<HELP_COMMAND> ::= PERFOFF
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the performance tracing off --> see HELP PERFON')
 END;
<HELP_COMMAND> ::= TRACEON
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the byte tracing mechanism on. This mechanism allows');
    WRITELN('the system developer to view the exact form of the');
    WRITELN('request and response blocks sent/received over the');
    WRITELN('communications line between the VAX and the DBP.')
 END;
<HELP_COMMAND> ::= TRACEOFF
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the byte tracing mechanism off --> see HELP TRACEON')
 END;
*==================================
* INPUT FROM DEVICE
*==================================
<INPUT> ::= INPUT
 BEGIN
    INPUT_FIRST := TRUE;
    INPUT_FILE := TRUE
 END;
*==================================
* LIST DATABASES
*==================================
<LIST_DBS> ::= LISTDBS
 BEGIN
    START;
    LIST_DATABASE(BLANK_IDENT,1,FALSE,TRUE);
    QUERY
 END;
*==================================
* LIST DATABASE
*==================================
<LIST_DB> ::= <LISTDBK> <LISTDB_REST>
 QUERY;
<LISTDBK> ::= LISTDB
 START;
<LISTDB_REST> ::= <ID>
```

```
 LIST_DATABASE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,TRUE,FALSE );
<LISTDB_REST> ::= ALL
 LIST_DATABASE( BLANK_IDENT,1,TRUE,TRUE );
*==================================
* LIST VIEWS
*==================================
<LIST_VIEWS> ::= <LIST_VIEWSK>
 BEGIN
    START;
    LIST_VIEWS;
    QUERY
 END;
<LIST_VIEWSK> ::= LISTVIEWS
 ;
*==================================
* LIST VIEW
*==================================
<LIST_VIEW> ::= <LIST_VIEWK> <WHICH_VIEW>
 BEGIN
    START;
    DESCRIBE_VIEW( ITEM1,ITEM1L,FALSE );
    QUERY
 END;
<LIST_VIEWK> ::= LISTVIEW
 ;
<WHICH_VIEW> ::= <ID>
 BEGIN
    ITEM1 := SSTACK[SP].IDNAME;
    ITEM1L:= SSTACK[SP].IDLEN
 END;
*==================================
* LOAD
*==================================
<LOAD> ::= <LOADK> <LV_REST>
 BEGIN
    END_CURSOR(1,TRUE);
    QUERY
 END;
<LOADK> ::= LOAD <ID>
 BEGIN
    START;
    SC_MODE := 1; (* RANDOM *)
    SC_DIRECTION := 0; (* FORWARD-ONLY *)
    SC_RETEST := 0;

    (* START A CURSOR FOR LOADING THIS VIEW *)

    (* ATTACH THE SOURCE VIEW, FIRST *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SSTACK[SP].IDNAME;
    ITEMS3L[1]:= SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    START_CURSOR( 1,SSTACK[MP+1].IDNAME,SSTACK[MP+1].IDLEN,
          SC_MODE,SC_DIRECTION,SC_RETEST )
 END;
<LV_REST> ::= <TUPLES>
 ;
<TUPLES> ::= <TUPLE>
 ;
<TUPLES> ::= <TUPLES> <TUPLE>
```

```
 ;
<TUPLE> ::= <START_TUPLE> <LV_ITEMS> <END_TUPLE>
 STORE( 1,INTEGRITY,IDCOUNT,ITEMS1,ITEMS1L );
<START_TUPLE> ::= [
 IDCOUNT := 0;
<END_TUPLE> ::= ]
 ;
<LV_ITEMS> ::= <LV_ITEM>
 ;
<LV_ITEMS> ::= <LV_ITEMS> <LV_ITEM>
 ;
<LV_ITEM> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[ IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<LV_ITEM> ::= <NO>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    NUM_TO_ASCII( SSTACK[SP].IVAL,ITEM1,ITEM1L );
    ITEMS1[ IDCOUNT ] := ITEM1;
    ITEMS1L[ IDCOUNT ] := ITEM1L
 END;
*==================================
* PERFORMANCE ON
*==================================
<PERFORM_COMMAND> ::= PERFON
 BEGIN
    WRITELN;
    WRITELN('Performance Monitoring is turned on.');
    PERFON
 END;
*==================================
* PERFORMANCE OFF
*==================================
<PERFORM_COMMAND> ::= PERFOFF
 BEGIN
    WRITELN;
    WRITELN('Performance Monitoring is turned off.');
    PERFOFF
 END;
*==================================
* TRACE ON
*==================================
<TRACE_COMMAND> ::= TRACEON
 BEGIN
    WRITELN;
    WRITELN('Trace is turned on.');
    TRACEON
 END;
*==================================
* TRACE OFF
*==================================
<TRACE_COMMAND> ::= TRACEOFF
 BEGIN
    WRITELN;
    WRITELN('Trace is turned off.');
    TRACEOFF
 END;
```

# APPENDIX C - A Sample DBPQL User Dialog

```
$
$ ql
  _TTB0: allocated
DBPQL : DBP Query Language Version 1.0
Ok

QL> help

The following commands are available :

CREATE   DELETE   DISPLAY   ECHO  INPUT   LIST
LOAD     PERFON   PERFOFF   TRACEON
TRACEOFF

General Notes :

1. To continue lines of input use a dash( - ) at the end
   of each line to be continued.
2. To exit DBPQL, just enter X at the prompt.

For a specific help, type HELP <command>

QL> help create

You can create several different entities :

CREATE DATABASE
CREATE RELATION
CREATE CONNECT VIEW
CREATE JOIN VIEW
CREATE PROJECT VIEW
CREATE SELECT VIEW
CREATE ORDER VIEW

To get more help on any one of these,
type HELP <one of the above lines>

QL> create database fem
Ok

QL> help create relation

You can define a new schema for a relation as follows:
CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA
  { <ITEMNAME> <TYPE1> <TYPE2> <LENGTH> }
  where : <TYPE1> = SIGNED_INT or INTEGER
                    UNSIGNED_INT
                    UNINTERPRET
                    ASCII
                    RECORD_PTR
                    STRING_PTR
          <TYPE2> = FIXED
                    VAR
          <LENGTH>= of item in bytes

You can also create a new relation which uses the
already defined schema of any given view :

CREATE RELATION <RELNAME> USING VIEW <OLD_VIEW>
```

```
QL> create relation beams in fem using schema -
QL> group integer fixed 4 -
QL> element integer fixed 4 -
QL> node1 integer fixed 4 -
QL> node2 integer fixed 4 -
QL> el-type ascii fixed 4 -
QL> nom-size ascii fixed 4 -
QL> material ascii fixed 8
Ok, View BEAMS has been created.
Ok

QL> listview beams
View : BEAMS
Underlying Relation : BEAMS
# of items =   7

GROUP                Signed Integer          4
ELEMENT              Signed Integer          4
NODE1                Signed Integer          4
NODE2                Signed Integer          4
EL-TYPE              ASCII Alphanumeric      4
NOM-SIZE             ASCII Alphanumeric      4
MATERIAL             ASCII Alphanumeric      8

QL>
QL>
QL> create relation nodes in fem using schema -
QL> node integer fixed 4 -
QL> xcoord integer fixed 4 -
QL> ycoord integer fixed 4 -
QL> zcoord integer fixed 4
Ok, View NODES has been created.
Ok

QL> listviews

List of Views :

BEAMS
FILE1
NODES
PEOPLE


QL> perfon
Performance Monitoring is turned on.

QL> load beams -
QL> [ 1 1 1 2 wf1 w8x8 aluminum ] -
QL> [ 1 2 3 4 i i3x2 titanium ] -
QL> [ 2 3 5 6 wf1 w8x8 graphite ]

Clock      27.44922
CPU       0.15000
Buffered I/O count      40
Direct I/O count       0
Page Fault count       0

Ok
```

```
QL> perfoff
Performance Monitoring is turned off.

QL> load nodes -
QL> [ 1 53 62 0 ] -
QL> [ 2 67 10 0 ] -
QL> [ 3 10 11 0 ] -
QL> [ 4 23 53 0 ] -
QL> [ 5 54 82 2 ] -
QL> [ 6 84 21 2 ]
Ok

QL> display beams

GROUP  ELEMENT  NODE1  NODE2  EL-TYPE  NOM-SIZE  MATERIAL
-----  -------  -----  -----  -------  --------  --------

1      1        1      2      WFL      W8X8      ALUMINUM
1      2        3      4      I        I3X2      TITANIUM
2      3        5      6      WFL      W8X8      GRAPHITE

QL> display nodes

NODE  XCOORD  YCOORD  ZCOORD
----  ------  ------  ------

1     53      62      0
2     67      10      0
3     10      11      0
4     23      53      0
5     54      82      2
6     84      21      2

QL> traceon
Trace is turned on.

QL> display nodes
== DBP REQUEST ==
# of bytes is     9
Byte Stream :

9A 05 4E 4F 44 45 53 FF 00                        ..NODES..


== DBP RESPONSE ==
# of bytes is   112
Byte Stream :

F9 02 9A 00 01 00 01 00 06 04 03 07 00 05 00 05  ................
4E 4F 44 45 53 05 4E 4F 44 45 53 00 00 00 01 01  NODES.NODES.....
02 04 00 03 00 01 04 04 4E 4F 44 45 00 05 4E 4F  ........NODE..NO
44 45 53 03 01 01 04 06 58 43 4F 4F 52 44 00 05  DES.....XCOORD..
4E 4F 44 45 53 03 02 01 04 06 59 43 4F 4F 52 44  NODES.....YCOORD
00 05 4E 4F 44 45 53 03 03 01 04 06 5A 43 4F 4F  ..NODES.....ZCOO
52 44 00 05 4E 4F 44 45 53 FF 0A 00 00 00 FF 00  RD..NODES.......


NODE  XCOORD  YCOORD  ZCOORD
```

```
== DBP REQUEST ==
# of bytes is    43
Byte Stream :

01 FF 06 FF 00 00 01 00 05 4E 4F 44 45 53 01 30  .........NODES.0
FF 00 02 01 01 05 4E 4F 44 45 53 01 20 FF 00 10  ......NODES. ...
01 01 FF 06 FF 00 03 FF 06 FF 00                 ...........


== DBP RESPONSE ==
# of bytes is   106
Byte Stream :

F1 01 01 01 31 02 35 33 02 36 32 01 30 FF 00 F1  ....1.53.62.0...
01 01 01 32 02 36 37 02 31 30 01 30 FF 00 F1 01  ...2.67.10.0....
01 01 33 02 31 30 02 31 31 01 30 FF 00 F1 01 01  ..3.10.11.0.....
01 34 02 32 33 02 35 33 01 30 FF 00 F1 01 01 01  .4.23.53.0......
35 02 35 34 02 38 32 01 32 FF 00 F1 01 01 01 36  5.54.82.2......6
02 38 34 02 32 31 01 32 FF 00 F6 08 10 00 00 64  .84.21.2.......d
00 00 1F 00 02 00 00 00 FF 00 0A 11 2E D9 0A 11  ................
00 00 1F 00 02 00 00 00 FF 00                    ..........


1      53      62      0
2      67      10      0
3      10      11      0
4      23      53      0
5      54      82      2
6      84      21      2

QL> traceoff
Trace is turned off.

QL> create join view j1 from beams node1 nodes node
Ok

     listview J1
     View : J1
     Underlying Relation : FEM
     # of items =  11

     GROUP                Signed Integer          4
     ELEMENT              Signed Integer          4
     NODE1                Signed Integer          4
     NODE2                Signed Integer          4
     EL-TYPE              ASCII Alphanumeric      4
     NOM-SIZE             ASCII Alphanumeric      4
     MATERIAL             ASCII Alphanumeric      8
     NODE                 Signed Integer          4
     X                    Signed Integer          4
     Y                    Signed Integer          4
     Z                    Signed Integer          4
```

```
create select view sel1 from beams where el-type= wfl
Ok

QL> display sel1

GROUP   ELEMENT   NODE1   NODE2   EL-TYPE   NOM-SIZE   MATERIAL
------  --------  ------  ------  --------  ---------  ----------
1       1         1       2       WFL       W8X8       ALUMINUM
2       3         5       6       WFL       W8X8       GRAPHITE

QL>
DBPQL - Goodbye.
$
```

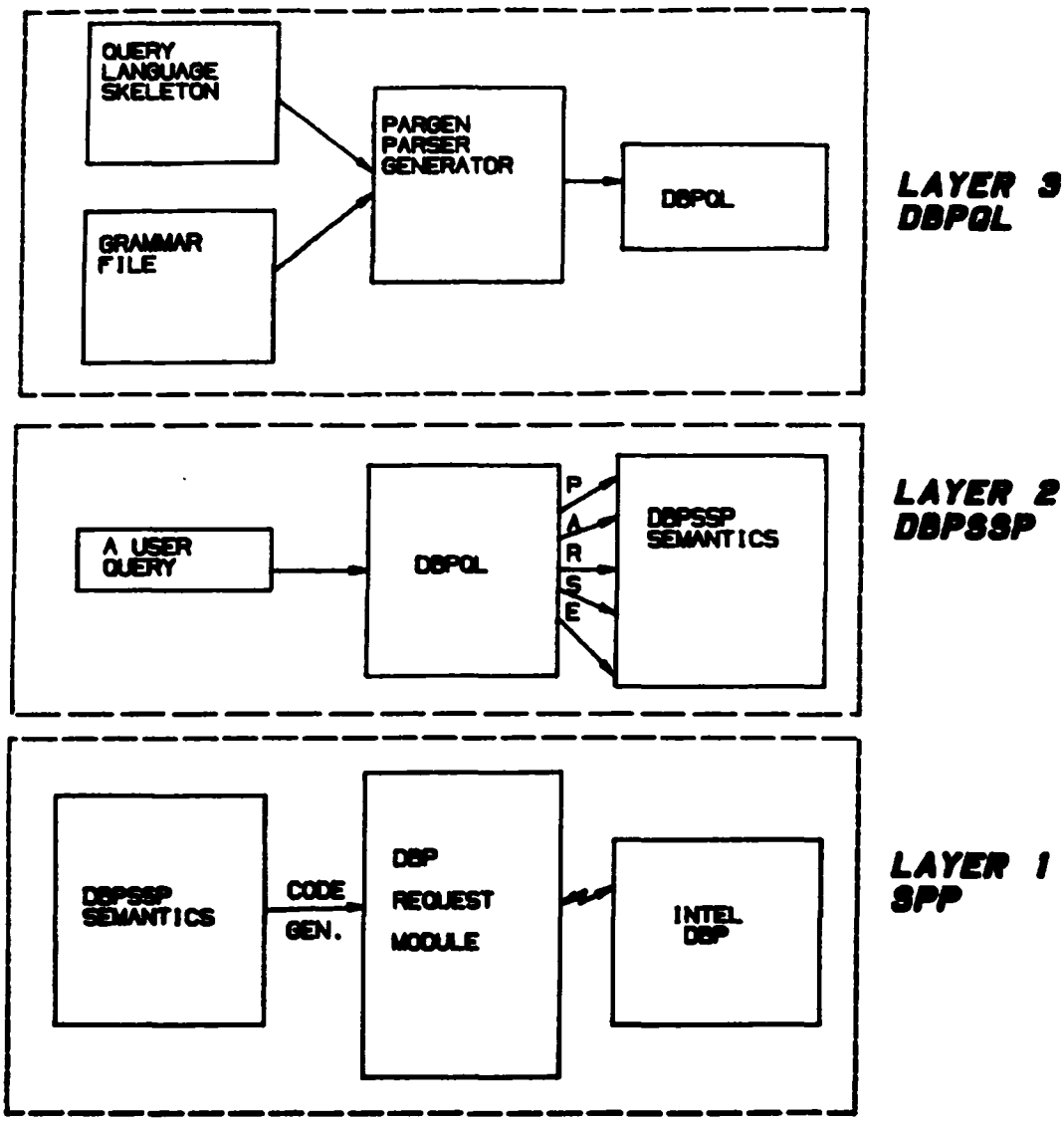



Figure 1. - HILDA: A general flow chart.

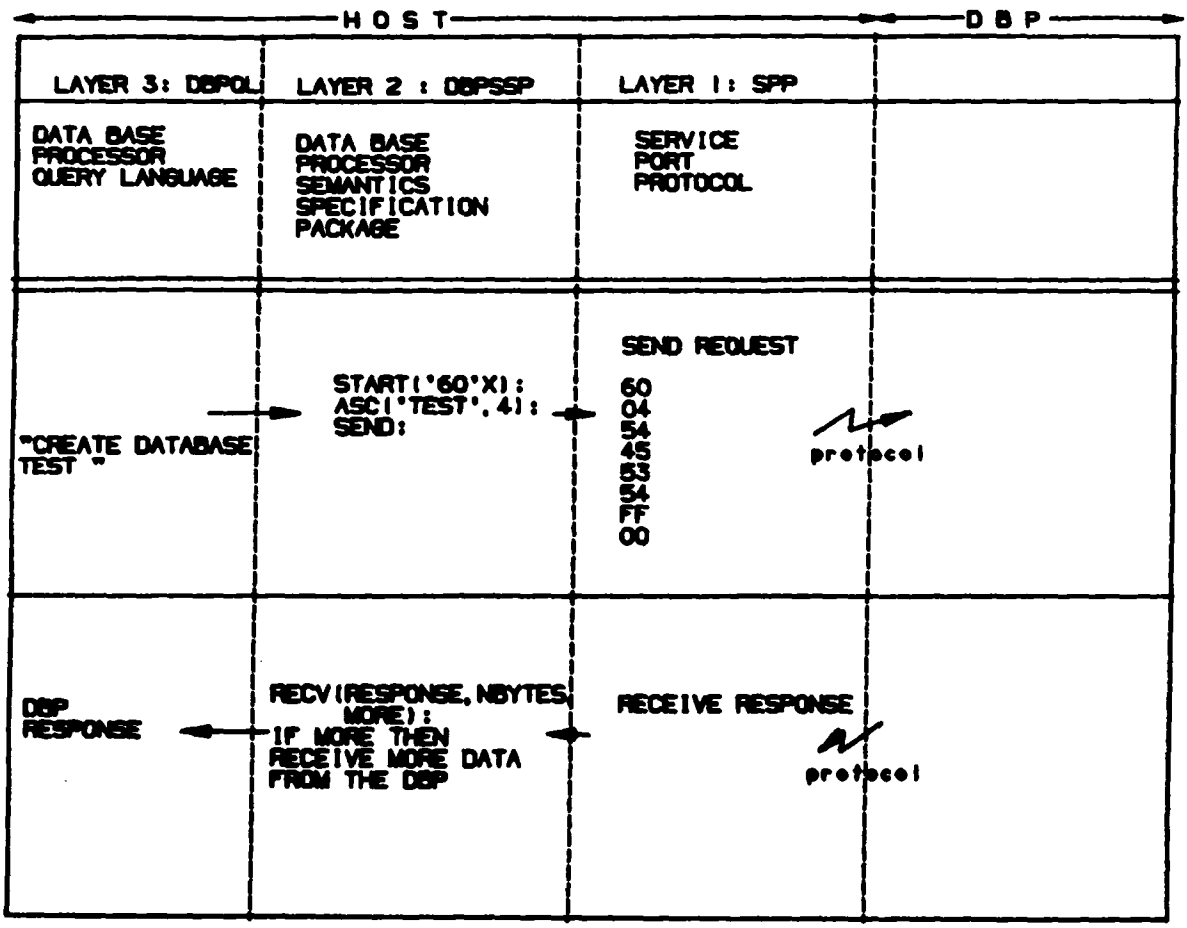



Figure 2. - HILDA: A sample query.

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| NASA CR-172191 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| DBPQL: A View-Oriented Query Language For The Intel Data Base Processor | July 1983 |
| | 6. Performing Organization Code |
| 7. Author(s) | 8. Performing Organization Report No. |
| Paul A. Fishwick | |
| | 10. Work Unit No. |
| 9. Performing Organization Name and Address | |
| Kentron Technical Center<br>3221 North Armistead Avenue<br>Hampton, VA 23666 | 11. Contract or Grant No.<br>NAS1-16000 |
| | 13. Type of Report and Period Covered |
| 12. Sponsoring Agency Name and Address | Contractor Report |
| National Aeronautics and Space Administration<br>Washington, DC 20546 | 14. Sponsoring Agency Code |

15. Supplementary Notes

Langley Technical Monitor: Timothy Rau
Final Report

16. Abstract

An interactive query language (DBPQL) for the Intel Data Base Processor (DBP) is defined. DBPQL includes a parser generator package which permits the analyst to easily create and manipulate the query statement syntax and semantics. The prototype language, DBPQL, includes trace and performance commands to aid the analyst when implementing new commands and analyzing the execution characteristics of the DBP. The DBPQL grammar file and associated key procedures are included as an appendix to this report.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| Database<br>Database Machine<br>HILDA<br>Query Language | Unclassified-Unlimited<br><br>Subject Category 62 |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 54 | A04 |



For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document