

Introduction to ADASQL

**WH&O International
Wellesley, Massachusetts**

Other Publication by WH&O International
Natural Study Guide
Advanced Natural Study Guide
Natural Editors
Natural Developers Handbook
Inside ADABAS
Natural Construct Study Guide
Natural Construct Applications Development Users Guide

7th Edition
Copyright ©2007 by WH&O International
All rights reserved.
Printed in the United States of America

It is illegal under Federal and International Copyright Laws and subject to statutory damages, to copy, reproduce or transmit all or part of this work in any form or by means electronically or mechanically, including, but not limited to photocopying, print, electronic information storage and retrieval systems on magnetic tape and/or disk or other media, without written permission from the publisher, WH&O International, Wellesley, MA 02482.

ADABAS, Natural, Predict, SuperNatural and Construct are registered trademarks of Software AG USA and Software AG of Darmstadt, Germany.
IBM, TSO, CICS, and DB2 are registered trademarks of International Business Machines.

Introduction to ADABAS SQL

ADABAS SQL allows access to ADABAS by the emerging standard for structured relational data languages. ADABAS SQL provides a standard interface making all language specific considerations transparent to the user; provides a standard interface across operating systems and teleprocessing environments. ADABAS SQL is closely integrated with PREDICT - Software AG's Data Dictionary which provides for centralized control of all corporate data.

The integration with PREDICT provides access to the definition of database files/tables and fields as well as PREDICT's active cross-referencing facility. Field definition attributes, extended field names and mnemonic field names, format, length, and synonyms are all extracted for PREDICT's repository. All structures are supported: elementary fields, multiple value fields (and appropriate count fields), periodic group structures and simple group structures. Cross references are maintained by program name, files and fields referenced, subroutines referenced, copycode members accessed, and also by programming language, date and time compiled.

Programs with ADABAS SQL statements are converted during execution of a preprocessor step prior to compilation (see Figure 13.1). The resulting source contains the ADABAS SQL statements, the generated code and the original host language source code. (The original source contains the ADABAS SQL statements but no generated code.) Any changes must be passed the preprocessor before recompilation.

ADABAS SQL is Software AG's response to the need for a common, user-friendly database language based on an already established standard. The principles of SQL were set forth in the early 1970's by E. F. Codd within a series of articles establishing the principles of relational databases. Typically, Software AG provides an easily to learn and easy to use interface for programmers and applications developers under ADABAS SQL. ADABAS SQL closely follows the American National Standards Institute (standards for the SQL language. ADABAS SQL supports COBOL, PL/1, Ada, and FORTRAN.

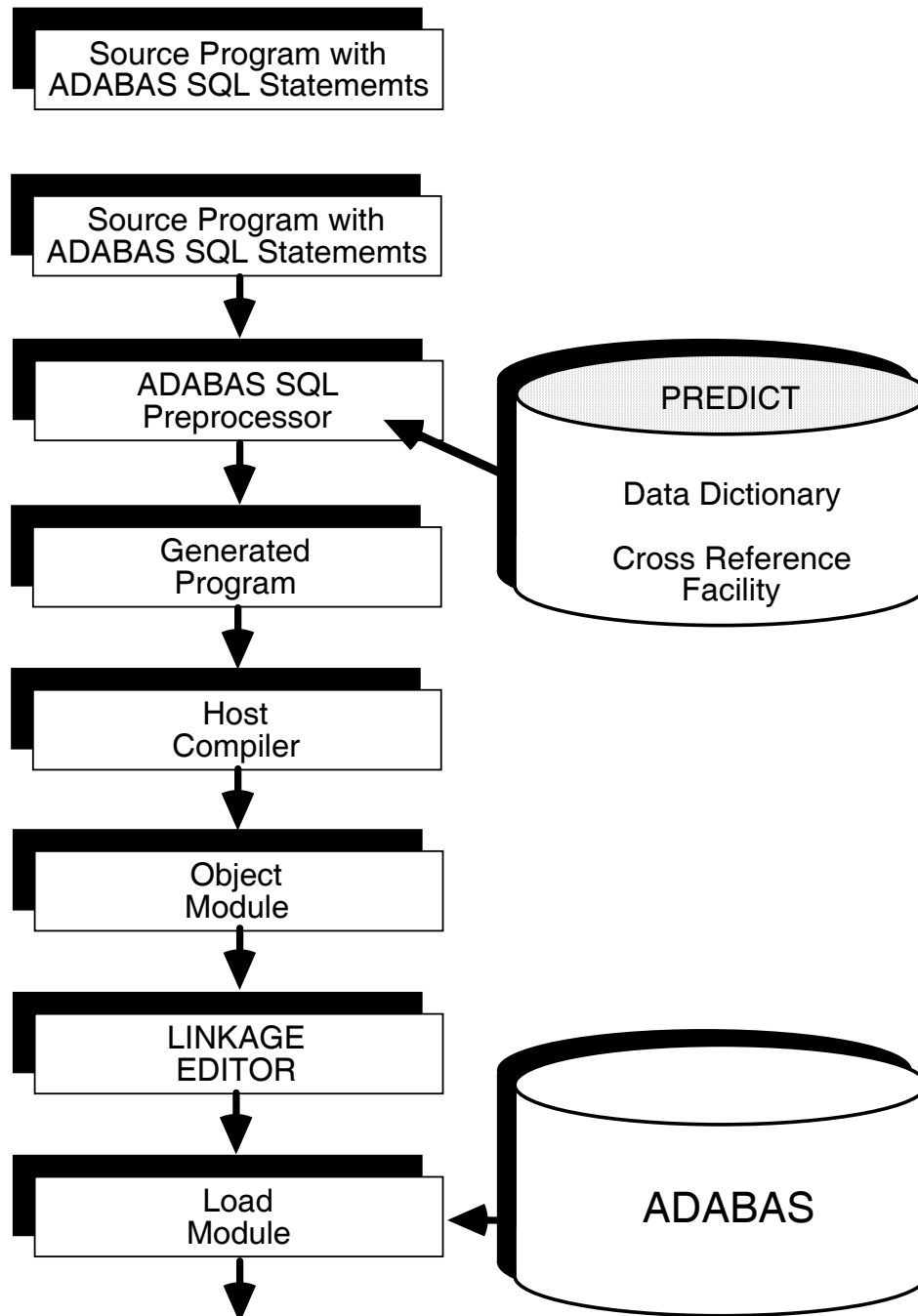


Figure 13.1 Implementation of ADABAS SQL

Basic Statement Syntax

The basic statement construct to ADABAS SQL is:

```
EXEC ADABAS
    statements
END-EXEC
```

The EXEC ADABAS statement must always be coded first and the END-EXEC must be coded last. Both must be the only statements on their respective line.

The statements coded between EXEC ADABAS and END-EXEC do have an established syntax and hierarchy. Currently, no more than 50 lines may be coded for each EXEC ADABAS statement. Only one ADABAS SQL statement is allowed within the EXEC ADABAS - END EXEC block.

Command Syntax;

[] parameters in brackets are optional
 { } at least one parameter within the braces must be specified
 ... parameter may be repeated
 parameter...4 means *parameter-1 parameter-2*
 etc. separated by a space whereas:
 parameter,...4 means *parameter-1, parameter-2,*
 etc. where the comma is required.
 aaa lower case parameter; variable information supplied by developer.
 AAA upper case parameter; keyword or reserved word.

ADABAS Buffer Overview

Record Buffer

The record buffer is an area of storage allocated by the program which is used by ADABAS to transfer information from the data base to the program or from the program to the data base.

If an *alias* is specified in the ADABAS SQL command which creates the Record Buffer, than that name is used as the level 1 qualifier. If an alias is not specified, then the *filename* or *viewname* is used as the level 1 qualifier.

A level 2 name is generated by ADABAS SQL which is for internal use only.

The data base field names are on level 3. The attributes for the field are taken from PREDICT, the Data Dictionary.

To refer to the data base fields from a COBOL program, the syntax would be:

field-name OF buffer-name

To refer to the data base fields from a PL/1 program, the syntax would be:

buffer-name.field-name

Record Buffer Extensions

ADABAS SQL appends three fields to each record in the Record Buffer. The fields are defined as follows:

ISN

A four byte binary field containing the Internal Sequence Number of the record found COBOL PIC 9(9) COMP; PL/1 FIXED BIN(31).

QUANTITY

A four byte binary field containing the number of records found for the specified search criteria COBOL PIC 9(9) COMP; PL/1 FIXED BIN(31). When used with a HISTOGRAM, QUANTITY contains the count field from the Inverted List, that is, the number of records on the data base for that specific descriptor value.

RESPONSE-CODE

A two byte binary field containing the response code for the execution of the ADABAS SQL command COBOL PIC 9(4) COMP; PL/1 FIXED BIN(15).

ISN Buffer

When a FIND statement returns more than one record, an ISN list is generated for all records which meet the search criteria.

The developer has the option of providing an ISN Buffer. The size of the buffer is determined by the ISNSIZE parameter specified in either the Globals or the individual ADABAS SQL statement (OPTIONS Clause).

If an ISN Buffer is provided and its size is large enough to hold the ISN List, then the list will be placed in the specified buffer.

If an ISN Buffer is not provided or if its size is not large enough to hold the ISN List, then the overflow ISNs will be placed on the ADABAS Work Data Set.

The ISNs are read from either the Work data set or the ISN Buffer and passed to the program one at a time. The process of transfer of these records from the Work Data Set or the ISN Buffer is transparent to the user.

General Statement Syntax

```
EXEC ADABAS
  ADABAS SQL-statement
END-EXEC
```

```
IDENTIFICATION DIVISION.
PROGRAM ID. MRSTMSQL.
:
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  EXEC ADABAS
    BEGIN DECLARE SECTION
  END-EXEC
  EXEC ADABAS
    SELECT PERSONNEL-ID, NAME, CITY
    FROM EMPLOYEES
    WHERE CITY = 'BOSTON'
  END-EXEC
:
PROCEDURE DIVISION
:
```

```

EXEC ADABAS
  statement-name
  [DECLARE cursor-name CURSOR [FOR]]
  [SELECT {select-list}
        { * }
  FROM file [alias],...
  [WHERE search-criteria]

  [
    [
      INDEXED={Y}
              {N}
      COND-NAME={Y}
                {N}
      HOLD [RETURN]
      PASSWORD={constant}
                {variable}
                { :variable }
      CIPHER={constant}
              {variable}
              { :variable }
      ISNSIZE=length {constant}
                   { :variable }
      SAVE
      SEQUENCE
      ISN=value
      PREFIX=prefix
      SUFFIX=suffix
      STATIC={Y}
             {N}
      MAXTIME=ss
      AUTOBID
      DBID=database-name
    ]
  ]

  [ORDER BY descriptor...3 [ {DESC [ENDING] } ] ]
  [ {ASC [ENDING] } ] ]
  [GROUP BY descriptor]
END-EXEC

```

Cursor-name may be thought of as view-name and *select-list* may be thought of as the list of fields to be retrieved. Aliases provide a means for more easily identifying view names by using a “synonym” approach. Filename aliases are coded immediately following the file/view name without any intervening commas: viewname1 alias, viewname2 alias.

You will find that statements involving JOINS, aliases play a significant role. In regards to this usage, the alias is actually defined in the FROM clause of the SELECT statement.

In the world of SQL outside the ADABAS environment, aliases take on many more meanings. In addition to file/table aliases, field/column-name aliases are also supported (this function only seems to aid in establishing headers for the column on reports).

You may also note the term “host variable” throughout Software AG manuals. These are simply user-defined variables and are used in the same manner. Any host variable used in an SQL statement must be prefixed with a colon, for example, if the password is contained in a user-defined variable, the password is supplied through: PASSWORD=:user-defined-variable-name.

Retrieval Types

statement-name	$\left\{ \begin{array}{l} \text{[FIND]} \\ \text{READ [PHYSICAL [SEQUENCE]]} \\ \text{READ LOGICAL} \\ \text{READ ISN} \\ \text{HISTOGRAM} \\ \text{SORT [ISN [LIST[S]]} \\ \text{COMPARE [ISN [LIST[S]]} \\ \text{FIND COUPLED} \end{array} \right\}$
----------------	--

If only the first record is to be retrieved, an ISN list will be produced and the data from the first record will be retrieved.

If all the records meeting the search criteria are to be processed, the Retrieval statement specifies the criteria, an OPEN statement produces the ISN list, the FETCH statement retrieves the records, and a CLOSE statement releases the ISN list.

FIND

Produces an ISN list for all the records which satisfy the given search criteria. If the keyword FOR is not coded in the DECLARE clause, then the data for the first record in the ISN list is retrieved.

If only the first descriptor in the file is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the FIND must be followed by the OPEN, FETCH, and CLOSE statements.

The only OPTIONS available for a FIND are HOLD, PASSWORD, CIPHER, ISNSIZE, and SAVE.

The list may be sorted in ORDER BY a descriptor.

The GROUP BY clause is not permitted for a FIND.

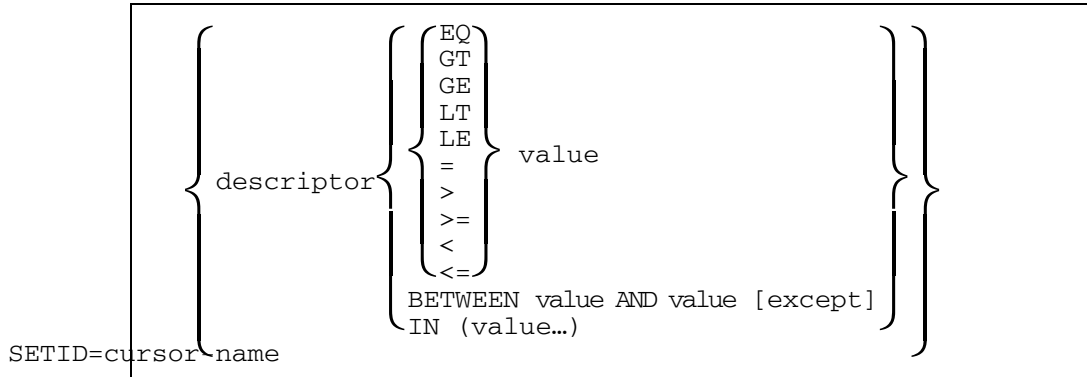
The search criteria for the FIND is specified as:

WHERE search-criteria

The WHERE clause specifies the criterion for the set of records to be retrieved. The format of the search criteria is:

```
= search-expression [ { AND } search-expression ] ...
```

The format for the search expression is:



The format for the exception clause is:

```
{ field NOT = value
  field NOT BETWEEN value AND value }
```

READ PHYSICAL SEQUENCE

Reads data from one or more records. Records are read in the order that they are physically stored in the data base.

If only the first record in the file is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the READ PHYSICAL SEQUENCE must be followed by the OPEN and FETCH statements.

Only one file may be specified in the FROM clause.

The WHERE, ORDER BY, and GROUP BY clauses may not be coded as part of a READ PHYSICAL SEQUENCE.

The only OPTIONS available for a READ PHYSICAL SEQUENCE are HOLD, PASSWORD, CIPHER, and ISN.

READ LOGICAL

Reads data from one or more records. Records are read in ascending logical order based on a given descriptor.

If only the first record in the file is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the READ LOGICAL must be followed by the OPEN and FETCH statements.

Only one file may be specified in the FROM clause. If alias is specified, it is used as the name of the Record Buffer. If it is not specified, the file name is used.

The only OPTIONS available for a READ LOGICAL are HOLD, PASSWORD, CIPHER, and ISN.

The descriptor specified in the ORDER BY clause must be the same as the descriptor in the WHERE clause. The search criteria is specified as:

WHERE field-name { ^{GE} >=} value

The WHERE clause specifies the starting point for the read of a descriptor.

Boolean logic may be performed on the ISN lists.

READ ISN

Reads data from a record based on a given ISN value.

Only one file may be specified in the FROM clause. If alias is specified, it is used as the name of the Record Buffer. If it is not specified, the name of the file is used.

The only OPTIONS available for a READ ISN is HOLD, PASSWORD, CIPHER, and SEQUENCE.

The search criteria is specified as:

WHERE { ISN= { constant :variable } CURRENT OF cursor-name1 }

The WHERE clause specifies the ISN of the record to be read. If the CURRENT option is specified, ADABAS SQL will use the ISN of a record that was found by the statement associated with that cursor name.

HISTOGRAM

The HISTOGRAM determines the values which are currently present for a descriptor. The number of records which contain that descriptor value is also available.

If only the first descriptor in the file is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the HISTOGRAM must be followed by the OPEN and FETCH statements.

Only one file may be specified in the FROM clause. If alias is specified, it is used as the name of the Record Buffer. If it is not specified, the name of the file is used.

The only OPTIONS available for a HISTOGRAM is PASSWORD.

The list is sorted by a descriptor specified with the ORDER BY clause.

The search criteria is specified as:

<pre>WHERE {descriptor BETWEEN value AND value} {descriptor {GE } value }</pre>

The WHERE clause specifies the range of values to be read for a descriptor.

SORT ISN LISTS

Sorts an ISN list previously produced by a FIND or a COMPARE. If the keyword FOR is not coded in the DECLARE clause, then the data for the first record in the ISN list is retrieved.

If only the first record in the file is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the SORT ISN LISTS must be followed by the OPEN, FETCH, and CLOSE statements.

Only one file may be specified in the FROM clause. If alias is specified, it is used as the name of the Record Buffer. If it is not specified, the name of the file is used.

The only OPTIONS available for a SORT ISN LISTS are HOLD, PASSWORD, CIPHER, ISNSIZE, and SAVE.

The list is sorted in ORDER BY up to three descriptors.

The GROUP BY clause may not be coded as part of a SORT ISN LISTS.

The search criteria is specified as:

<pre>WHERE CURSOR = cursor-name</pre>

The WHERE clause specifies the cursor name coded in the DECLARE clause of the statement that created the ISN list being used in the SORT.

COMPARE ISN LISTS

Produces an ISN list which is a combination of two other previously produced ISN lists.

If only the first record in the file is needed, omit the DECLARE clause. If multiple records are to be processed, the COMPARE ISN LISTS must be followed by the OPEN and FETCH statements.

Only one file may be specified in the FROM clause.

The only OPTIONS available for a COMPARE ISN LISTS are HOLD, PASSWORD, CIPHER, ISNSIZE, and SAVE.

The search criteria is specified as:

<pre>WHERE CURSOR = cursor-name { AND OR AND NOT } CURSOR = cursor- name</pre>
--

The WHERE clause specifies the cursor names representing the ISN lists which were created in the previous FIND or COMPARE statements.

Boolean logic may be performed on the ISN lists.

FIND COUPLED

Locates a specific record in a secondary file based on the FIND of a record in the primary file. If the keyword FOR is not coded in the DECLARE clause, then the data for the first record in the ISN list is retrieved.

If only the first record meeting the search criteria is needed, the DECLARE clause may be omitted. If multiple records are to be processed, the FIND COUPLED must be followed by the OPEN, FETCH, and CLOSE statements.

Only two files may be specified in the FROM clause.

The ORDER BY and GROUP BY clauses may not be coded as part of a FIND COUPLED.

The only OPTIONS available for a FIND COUPLED are HOLD, PASSWORD, CIPHER, and ISNSIZE, and SAVE.

The search criteria is specified as follows:

<pre>WHERE ISN = value</pre>

The WHERE clause specifies the ISN of the record in the secondary file to which the record in the primary file is coupled.

DECLARE Clause

<pre>[DECLARE cursor-name CURSOR [FOR]]</pre>

Cursor-name identifies the current statement so that it can subsequently be referred to by other statements specifying the *cursor-name*. In reality, cursors define a (DB2) table of results or a (ADABAS) result list. Records are made available to the program one at a time and the cursor provides a pointer to the current record (top down reading). Each time a record is read

(FETCH in SQL), the pointer (cursor) is moved down the table and so on.

Cursor-name generates an ADABAS Command-ID.

Cursor-name may be up to 4 characters in length.

If multiple records are to be processed, the keyword FOR must be specified. This statement must then be used in conjunction with the OPEN and FETCH statements.

If a single record is to be processed, the DECLARE clause may be omitted.

SELECT Clause

[SELECT { ^{field-list} * }]
--

Specifies which fields are to be retrieved. The field names in the select-list must be specified by the name defined in the data dictionary. Fields not in the select-list are not placed in the Record Buffer.

```
SELECT NAME, FIRST-NAME, MIDDLE-I
```

Redefined fields and phonetic descriptors cannot be included in the select-list. If a name in select-list is a Group field name, ADABAS SQL automatically generates the definitions for the lower-level fields.

```
SELECT FULL-NAME
```

If a Multiple Value Field is specified in the select-list, the number of occurrences is taken from the dictionary. If that value is zero, the number of occurrences in ADABAS SQL is set to 191. The number of occurrences may be explicitly specified in parentheses following the field name. Optionally, the word COUNT may be specified in parentheses following the field name indicating that the count of the actual number of occurrences should be calculated and included in the Record Buffer. This counter is a two byte binary field. The generated name is "C-" or "C_" followed by the multi-value field name.

```
SELECT ADDRESS-LINE
SELECT ADDRESS-LINE(1)
SELECT ADDRESS-LINE(:variable1)
SELECT ADDRESS-LINE(1-3)
SELECT ADDRESS-LINE(LAST)
SELECT ADDRESS-LINE(1-LAST)
```

If a Periodic group is specified in the select-list, ADABAS SQL automatically generates the definitions for all the lower-level fields. The number of occurrences is taken from the dictionary. If that value is zero, the number of occurrences in ADABAS SQL is set to 191. The number of occurrences may be explicitly specified in parentheses following the field name. Optionally, the word COUNT may be specified in parentheses following the field name indicating that the count of the actual number of occurrences should be calculated and included in the Record Buffer. This counter is a two byte binary field. The generated name is "C-" or "C_" followed by the Periodic Group.

```
SELECT SALARY(1)
SELECT SALARY(:variable)
SELECT SALARY(1-12)
SELECT SALARY(C-COUNT)
```

Don't code the element, it is not the definition of the multiple occurring group, instead:

```
SELECT INCOME(C-COUNT)
```

If fields within a Periodic Group are requested, the COUNT field is not available.

If a Period Group contains a Multiple Value Field, it should be requested separately.

```
SELECT BONUS (1(1))
SELECT BONUS (1(1-3))
SELECT BONUS (1-3(1))
SELECT BONUS (1-2(1-2))
SELECT BONUS (:variable(1))
SELECT BONUS (:variable(1-3))
SELECT BONUS (1(:variable))
SELECT BONUS (1-2(:variable))
SELECT BONUS (:variable(:variable))
SELECT BONUS (LAST)
SELECT BONUS (LAST(LAST))
SELECT BONUS (1(1-LAST))
```

If an "*" is specified, all fields in the view will be selected.

```
SELECT *
```

If the SELECT clause is omitted, no records are processed. However, other functions may still be performed.

FROM Clause

```
FROM file [alias],...5
```

If "alias" is specified, it is used as the name of the Record Buffer, otherwise, the File name is used. An alias is required if two or more ADABAS SQL statements refer to the same file. The alias is then subsequently used as a high-level qualifier.

WHERE Clause

```
[ WHERE search-criteria ]
```

This clause specifies the search criteria for the requested records. This is described for each type of Retrieval statement in the previous pages under each Retrieval name.

OPTIONS Clause

INDEXED is available for COBOL program only. MU and PEs are generated with the INDEXED BY keywords. If there is no specification on PREDICT then the name of the MU or PE is prefixed with "I-".

Condition names, **COND-NAME**, defined to PREDICT may be generated into record buffers of COBOL programs. These are defined as level-88 entries if COND-NAME=Y.

If **HOLD** is specified, the retrieved record is placed in hold status and cannot be updated or deleted by another user. If a record is to be updated or deleted, it must previously have been placed in hold status.

PASSWORD must be specified if the file is secured through ADABAS Security unless the password is specified in the CONNECT statement. The value specified is the ADABAS password. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a host variable is to place a ":" prior to the variable name in the PASSWORD clause.

CIPHER must be specified if the file has been encrypted. The value specified is for the cipher code. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a host variable is to place a ":" prior to the variable name in the CIPHER clause.

The **ISNSIZE** indicates the maximum number of ISNs which can be stored in the ISN Buffer. If the number of ISNs retrieved exceeds limit, the excess will automatically be retrieved as needed transparent to the user. If this option is not coded, the

ISN Buffer size is taken from the OPTIONS (global parameter) statement. If neither or these are coded, an ISN Buffer is not allocated. This must be the case if the file is protected by Security By Value. This can be specified for a COMPARE, FIND, FIND COUPLED, or SORT statement.

OPTIONS	INDEXED= { Y N } COND-NAME= { Y N } HOLD= [RETURN] PASSWORD= { constant variable :variable } CIPHER= { constant variable :variable } ISNSIZE=length { constant :variable } SAVE SEQUENCE ISN=value PREFIX=prefix SUFFIX=suffix STATIC= { Y N } MAXTIME=ss AUTODBDID DBID=database-name
---------	--

The **SAVE** option is used to retain the ISN list. The saved list will be deleted when a subsequent FIND with the same name is encountered; when a CLOSE is executed; or when a transaction time limit or non-activity time limit is reached. This can be specified for a COMPARE, FIND, or SORT statement.

If the **SEQUENCE** option is specified, the record with the specified ISN (or next higher ISN) will be retrieved. The ISN of the retrieved record is placed in the ISN field in the Record Buffer. If the end of file is encountered, ADACODE will indicate the end-of-file status. This can be specified for a READ ISN statement only.

The **ISN** option can be specified for a READ PHYSICAL SEQUENCE or READ LOGICAL statement. For a READ PHYSICAL SEQUENCE, it indicates the ISN for the first record to be read. If a record does not exist for that ISN, the record with the next higher ISN will be retrieved. For a READ LOGICAL, it indicates the ISN for the first record to be read from the set which satisfies the WHERE clause. If the value is specified as a host variable, the syntax is to place a ":" prior to the variable name in the ISN clause.

ORDER BY Clause

```
[ ORDER BY descriptor...3 { DESC [ENDING]
  ASC [ENDING] } ]
```

Specifies the order by which records are to be returned for a FIND, HISTOGRAM, READ LOGICAL and SORT. The FIND and SORT may specify up to three descriptors in either ascending or descending sequence.

The READ LOGICAL may only specify one descriptor and is always specified in ascending sequence.

Although this clause may be specified in a HISTOGRAM, it has no effect.

The descriptor(s) may not be within a Periodic Group and may not be a Phonetic descriptor.

Ascending is the default sequence.

GROUP BY Clause

```
[ GROUP BY descriptor ]
```

Specifies the descriptor for which values are to be retrieved for a HISTOGRAM. If a WHERE clause is also specified, the descriptor in the GROUP BY clause must be the same descriptor in the WHERE clause.

Multi-Record Processing Statements**OPEN Statement**

```
EXEC ADABAS
  OPEN cursor-name
END-EXEC
```

An OPEN must be executed after the Retrieval statement and prior to the FETCH of any data. It processes the parameters specified in the WHERE clause of the statement referenced by the cursor-name.

FETCH Statement

```
EXEC ADABAS
  FETCH cursor-name
END-EXEC
```

FETCH retrieves the data values from the data base and places the data into the Record Buffer. An OPEN statement must previously have been executed.

Each subsequent FETCH statement reads the next record into the Record Buffer until all records have been exhausted.

CLOSE Statement

```
EXEC ADABAS
  CLOSE [ cursor-name ]
END-EXEC
```

This statement releases the ISN list as well as performing other clean-up duties. It is issued after a FIND, SORT, COMPARE, or FIND COUPLED. It may optionally be issued after a READ LOGICAL, READ PHYSICAL SEQUENCE, or HISTOGRAM.

Data Base Modification Statements

Logical Transaction Processing

Logical Transactions are defined as changes to a single data base record, multiple data base records or records spanning multiple files. The Logical Unit of Work is defined by the user.

DELETE Statement

```
EXEC ADABAS
  DELETE
  [ DECLARE cursor-name CURSOR ]
  FROM file
  WHERE { ISN=value
         CURRENT OF cursor-name }
  [ OPTIONS [ PASSWORD= { constant
                       :variable } ]
          [ CIPHER= { constant
                     :variable } ]
          STATUS ] ]
END-EXEC
```

A DELETE statement is executed in order to delete a record from the data base. The record to be deleted must previously have been read and placed in hold status.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

Cursor-name identifies the current statement so that it can subsequently be referenced to by other statements specifying the same Cursor-name. Cursor-name may be up to 4 characters in length.

FROM Clause

```
FROM filename
```

This specifies the name of the file from which data is to be deleted.

WHERE Clause

```
WHERE { ISN=value  
CURRENT OF cursor-name }
```

The WHERE clause indicates the ISN of the record to be deleted. If the ISN is known, it may be specified directly. If it is not known, the cursor name may be specified which references the retrieval statement.

OPTIONS Clause

```
[ OPTIONS [ PASSWORD= { constant  
:variable } ]  
[ CIPHER= { constant  
:variable } ]  
[ STATUS ] ]
```

PASSWORD must be specified if the file is secured through ADABAS Security unless the password is specified in the CONNECT statement. The value specified is the ADABAS password. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a variable is to place a ":" prior to the variable name in the PASSWORD clause.

CIPHER must be specified if the file has been encrypted. The value specified is for the cipher code. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a variable is to place a ":" prior to the variable name in the CIPHER clause.

If the STATUS option is specified, data protection information is written to the Data Protection Log.

INSERT Statement

```

EXEC ADABAS
  INSERT INTO file [alias]
  [ DECLARE cursor-name CURSOR ]
  [ WHERE ISN=value, ]
  {
    SET {
      {field
        {constant
         :variable}
      }
    }
    {
      field,... VALUES ( {constant
                          :variable} ,... )
    }
  }
  [
    OPTIONS [
      PASSWORD={
        constant
        variable
        :variable
      }
      CIPHER={
        constant
        :variable
      }
      PREFIX=prefix
      SUFFIX=suffix
      STATUS
    ]
  ]
END-EXEC

```

An INSERT statement is executed in order to insert a new record into the data base.

INTO Clause

```
INTO file [alias]
```

This clause specifies the name of the file INTO which data is to be inserted. Alias specifies the name of the Record Buffer. If the alias is not supplied, the name of the file is used as the name of the Record Buffer.

DECLARE Clause

```
[ DECLARE cursor-name CURSOR ]
```

Cursor-name identifies the current statement so that it can subsequently be referred to by other statements specifying the same Cursor-name. Cursor-name may be up to 4 characters in length.

WHERE Clause

$\left[\text{WHERE } \left\{ \begin{array}{l} \text{ISN=value} \\ \text{CURRENT OF cursor-name} \end{array} \right\} \right]$
--

The WHERE clause indicates the ISN of the record to be added. If the WHERE clause is omitted, ADABAS will generate the Internal Sequence Number.

The use of "WHERE CURRENT OF..." provides better buffer usage for INSERT and UPDATE statements. If the fields are all the same, avoid the use of the SET option for improved Record Buffer efficiency.

SET Clause

$\left\{ \begin{array}{l} \text{SET } \left\{ \begin{array}{l} \text{field} \\ \text{field} = \left\{ \begin{array}{l} \text{constant} \\ \text{variable} \end{array} \right\} \end{array} \right\} \\ \text{field,... VALUES } \left(\left\{ \begin{array}{l} \text{constant} \\ \text{variable} \end{array} \right\} , \dots \right) \end{array} \right\}$

The SET clause indicates the fields to be inserted as part of the record to be added and optionally initializes these fields.

The field = constant/variable is used to initialize the value of the field. The value assigned may be a constant or literal or the value of a variable defined in the host language program. Variable name usage:

$\left\{ \begin{array}{l} \text{:variable [(index)]} \\ \text{:root.variable [(index)]} \\ \text{:variable [(index)] } \left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \text{root} \end{array} \right\}$	<i>PL/1-type</i>
	<i>COBOL-type</i>

Index is optional and may be assigned as:

$\left[\left(\begin{array}{l} \text{integer constant} \\ \text{:variable} \end{array} \right) \right]$
--

OPTIONS Clause

[OPTIONS	[PASSWORD= { constant :variable } CIPHER= { constant :variable } PREFIX=prefix SUFFIX=suffix STATUS]]
---	---------	---	--	---	---

PASSWORD must be specified if the file is secured through ADABAS Security unless the password is specified in the CONNECT statement. The value specified is the ADABAS password. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a host variable is to place a ":" prior to the variable name in the PASSWORD clause.

CIPHER must be specified if the file has been encrypted. The value specified is for the cipher code. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a variable is to place a ":" prior to the variable name in the CIPHER clause.

If the STATUS option is specified, data protection information is written to the Data Protection Log.

If PREFIX is specified, field names generated in the Record Buffer will be prefixed with the value coded.

If SUFFIX is specified, field names generated in the Record Buffer will be suffixed with the value coded.

PREFIX or SUFFIX may be coded in an OPTIONS Clause, global OPTIONS parameter or the PREDICT data dictionary.

UPDATE Statement

```

EXEC ADABAS
  UPDATE file [alias]
  [ DECLARE cursor-name CURSOR ]
  FROM file
  WHERE { ISN=value
         CURRENT OF cursor-name }
  { SET { field
        field = { constant }
              { :variable }
        field,... VALUES ( { constant } ,... )
      }
  [ OPTIONS [ PASSWORD= { constant
                    :variable }
            CIPHER= { constant
                    :variable }
            PREFIX=prefix
            SUFFIX=suffix
            STATUS ] ] ]
END-EXEC

```

An UPDATE statement is executed in order to modify a record in the data base. The record to be updated must previously have been read and placed in hold status.

FILE Clause

```
UPDATE file [alias]
```

This clause specifies the name of the file in which data is to be UPDATED. Alias specifies the name of the Record Buffer. If the alias is not supplied, the name of the file is used as the name of the Record Buffer.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

Cursor-name identifies the current statement so that it can subsequently be referred to by other statements specifying the same Cursor-name. Cursor-name may be up to 4 characters in length.

WHERE Clause

```
WHERE { ISN=value
        CURRENT OF cursor-name }
```

The WHERE clause indicates the ISN of the record to be updated. If the ISN is known, it may be specified directly. If it is not known, the cursor name may be specified which relates back to the retrieval statement.

The use of "WHERE CURRENT OF..." provides better buffer usage for INSERT and UPDATE statements. If the fields are all the same, avoid the use of the SET option for improved Record Buffer efficiency.

SET Clause

```
{ SET { field
        field = { constant
                 :variable }
        field,... VALUES ( { constant
                             :variable } ,... ) }
```

The SET clause indicates the fields to be updated as part of the record to be updated and optionally supplies the new values for these fields.

The field = constant / variable is used to initialize the value of the field. The value assigned may be a constant or literal or the value of a variable defined in the host language program. Variable name usage:

```
{ :variable [(index)]
  :root.variable [(index)] } PL/1-type
{ :variable [(index)] { OF
                       IN } root } COBOL-type
```

Index is optional and may be assigned as:

```
[ ( integer constant )
  ( :variable ) ]
```

OPTIONS Clause

<pre> [OPTIONS [PASSWORD= { constant :variable } CIPHER= { constant :variable } PREFIX=prefix SUFFIX=suffix STATUS]] </pre>

PASSWORD must be specified if the file is secured through ADABAS Security unless the password is specified in the CONNECT statement. The value specified is the ADABAS password. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a host variable is to place a ":" prior to the variable name in the PASSWORD clause.

CIPHER must be specified if the file has been encrypted. The value specified is for the cipher code. If the value is specified as a constant, it will print in a listing. If this is not desirable, it may be specified as a variable. The syntax for specifying a variable is to place a ":" prior to the variable name in the CIPHER clause.

If the STATUS option is specified, data protection information is written to the Data Protection Log.

If PREFIX is specified, field names generated in the Record Buffer will be prefixed with the value coded.

If SUFFIX is specified, field names generated in the Record Buffer will be suffixed with the value coded.

PREFIX or SUFFIX may be coded in an OPTIONS Clause, global OPTIONS parameter or the PREDICT data dictionary.

COMMIT WORK Statement

<pre> EXEC ADABAS COMMIT WORK [USERDATA = value] END-EXEC </pre>
--

A COMMIT WORK statement is executed in order to commit modifications to the data base. It also indicates the end of one logical transaction and the beginning of another.

User data may optionally be written to the ADABAS System File. This data may later be retrieved by a CONNECT or a READ USERDATA statement. If user data is to be written, a CONNECT statement must previously have been issued specifying a valid user ID.

ROLLBACK WORK Statement

```
EXEC ADABAS
  ROLLBACK WORK [WITHOUT filename]
END-EXEC
```

A ROLLBACK WORK statement is executed in order to back out modifications to the data base:

- since the beginning of the session, or
- since the last COMMIT WORK, or
- to the last ROLLBACK WORK.

If the WITHOUT clause is coded, the filename specified will be excluded from the back out process.

READ USERDATA Statement

```
EXEC ADABAS
  READ USERDATA
  INTO variable
  [USERID = value]
END-EXEC
```

A READ USERDATA statement is executed in order to read the user data stored during a COMMIT WORK, CHECKPOINT, or DBCLOSE. Currently, the amount of data is limited to 500 bytes.

The INTO clause specifies where the User Data is to be placed. The USERID clause specifies the User ID associated with the User Data on the ADABAS System File.

CHECKPOINT Statement

```
EXEC ADABAS
  CHECKPOINT { USER=value
              SYNC2
              SYNC3 [USERDATA=value] }
END-EXEC
```

A CHECKPOINT statement is used to synchronize all programs which access a file cluster and to perform the checkpoint.

If USER is specified, an ADABAS "C1" command is generated. A four-byte "value" indicates the checkpoint code. Value may also be a variable in which case the variable name must be preceded by ":". If SYNC2 is specified, an ADABAS "C2" command is generated. If SYNC3 is specified, and ADABAS "C3" command is generated.

Additional ADABAS SQL Statements

BEGIN Statement

```
EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
```

The BEGIN Statement must be the first ADABAS SQL statement in every program. It must be within the Data Division in a COBOL program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXBEGIN.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
:
```

CONNECT Statement

```
EXEC ADABAS
  CONNECT
    [userid]
    [WITH password]
    [ACC=file,...]
    [UPD=file,...]
    [EXU=file,...]
    [CLU=file,...]
    [AND USERDATA INTO variable]
    [OPTIONS [DBID=database-name]
      [MAXISN=value]
      [MAXHOLD=value]
      [MAXCID=value]
      [MAXTIME]
      [TT=value]
      [TNA=value]]
END-EXEC
```

The CONNECT statement indicates which files are to be accessed and specifies the ADABAS security levels which are required for that access. If the CONNECT statement is omitted, the program automatically runs in ET user mode. If it is specified, it must be specified in the main program.

The userid specifies the User Id for this session. It must be specified if User Data is to be created or accessed.

A password may be globally specified in the CONNECT statement. ADABAS SQL will then pass this password on to all ADABAS commands. If a local password is subsequently specified, it overrides the global password.

The ACC clause specifies the list of access only files to be accessed by this program. The UPD clause specifies the list of files to be modified by an ET user program. The CLU clause specifies the list of cluster files to be accessed by this program. The EXU clause specifies the list of files to be used in exclusive user mode.

The USERDATA clause allows for retrieval of User Data previously stored by a DBCLOSE, CHECKPOINT, or COMMIT WORK statement.

OPTIONS Clause

DBID allows access to more than one database; MAXISN specifies the number of ISNs to be stored internally; MAXHOLD specifies the maximum number of ISNs any user may place in hold status; MAXCID specifies the maximum number of command IDs active; MAXTIME sets a time limit for S-type commands; TT sets a transaction time limit; TNA sets a non-activity time limit.

COPY Statement

```
EXEC ADABAS
  COPY [FILE=]file-name [[MEMBER=]member-name]
END-EXEC
```

The COPY statement indicates that file layouts generated by PREDICT are to be included as either COBOL copy code or PL/1 include code. The file-name must be specified as it is in PREDICT. The member-name must be specified if more than one file layout has been generated for the file-name.

DBCLOSE Statement

```
EXEC ADABAS
  DBCLOSE [USERDATA=value]
  [OPTIONS DBID=database-name]
END-EXEC
```

This statement indicates the completion of ADABAS processing. The USERDATA clause may be used to store user data on the ADABAS System File. It has been suggested that each database opened should be closed separately. DBID should specify a database-name defined in the data dictionary.

HOLD Statement

```
EXEC ADABAS
  HOLD cursor-name
  [OPTIONS RETURN]
END-EXEC
```

The HOLD statement places a record in hold status. A record must be placed in hold status prior to executing an UPDATE or a DELETE. The cursor-name identifies the statement that retrieved the record.

If the OPTIONS RETURN clause is coded and the record is already in hold status by another user, a response code of "145" will be returned to the program. If OPTIONS RETURN is not specified, the program will be placed in a wait state until the record is released.

RELEASE Statement

```
EXEC ADABAS
  RELEASE cursor-name
  [FOR [FORMAT] [SEQ] [LIST]]
END-EXEC
```

The RELEASE statement releases an ADABAS Command ID. FORMAT releases the Command ID from the Format Buffer Pool. SEQ releases the Command ID from the Table of Sequential Commands. LIST releases the Command ID from the Table of ISN Lists.

RELEASE ISN Statement

```
EXEC ADABAS
  RELEASE ISN cursor-name
END-EXEC
```

The RELEASE ISN statement releases a record from hold status. This statement should not be used by an ET user if a modification has been made to the file previously.

RESTORE Statement

```
EXEC ADABAS
  RESTORE cursor-name :variable
END-EXEC
```

The RESTORE statement will move the ADABAS Control Block for the cursor-name to the user-variable to be used in the preparation of pseudo-conversational programs under CICS. The programmer must then read this data area from CICS Temp Storage prior to executing the RESTORE.

SAVE Statement

```
EXEC ADABAS
  SAVE cursor-name :variable
END-EXEC
```

The SAVE statement will move the ADABAS Control Block for the cursor-name to the user-variable, which must be an 80 byte field, to be used in the preparation of pseudo-conversational programs under CICS. The programmer must then write this data area to CICS Temp Storage.

TRACE Statement

```
EXEC ADABAS
  TRACE { ON
        OFF }
END-EXEC
```

The TRACE statement is used to trace the execution of all ADABAS SQL commands. The SET GLOBAL option MODE TRACE must be on. If the TRACE is turned ON, the ADABAS SQL preprocessor will generate code to print the trace information. The TRACE information may also be saved to a dataset with a specified DDNAME.

WRITE TO LOG Statement

```
EXEC ADABAS
  WRITE TO LOG
  USERDATA=:value
  [STATUS]
END-EXEC
```

The WRITE TO LOG statement is used to write data to the ADABAS Protection Log. The data to be written to the log must be specified as a variable. The variable name must be preceded by a ":". The STATUS option causes the data to be physically written to the Data Protection Log as soon as the statement is executed. Available in ADABAS Version 4 only.

Internal Error Checking

ADABAS SQL automatically calls an error routine if the response code from an ADABAS SQL command is non-zero. The default name for the error routine is RESPINT for Cobol and PL/1.

The response code is stored in the response code field appended to every record in the Record Buffer.

For every non-zero response code (except "3"), the error routine prints out an error message, the contents of the ADABAS Control Block, and the line number in the source program of the statement causing the error. It then terminates execution of the ADABAS SQL program.

If an action other than the one specified above is required, the global parameter ABORT may be specified to invoke a user-supplied error routine. For more information on this parameter, refer to the Global Parameters section.

Common Response Codes

- 001 The ISN list is too large to be sorted.
- 003 End-of-file condition detected.
- 005 Used to synchronize programs before taking a checkpoint when using file clusters.
- 009 Transaction has been backed out due to a time-out.
- 017 Invalid file number.
- 019 An attempt was made to update a file that was open for access only.
- 048 The User-Id specified in the CONNECT statement is already in user; or the mode of usage specified for the file conflicts with its current usage.
- 113 A READ ISN was executed but no record with that ISN was found; or the READ ISN failed the ADABAS Security-By-Value check; or an INSERT was specified for a record with an ISN which already exists.
- 144 An UPDATE or DELETE command was issued but the record was not previously placed in hold status.
- 145 An attempt was made to hold a record which was already in hold status by another user.
- 148 ADABAS nucleus was not available.
- 198 An INSERT command was issued on a record with a unique descriptor where the descriptor value for the record to be inserted already exists on the file.

On-line Processing

COM-PLETE

ADABAS SQL programs which will run under the control of the COM-PLETE Teleprocessing Monitor should be coded exactly the same as for a batch program.

The COM-PLETE Utility USCHC should be set so that the output produced by a DISPLAY statement will be sent to the User's terminal.

CICS

ADABAS SQL programs which will run under the control of the CICS Teleprocessing Monitor will be quasi-reentrant. CICS pseudo-conversational mode is not supported.

The Transaction Work Area (TWA) is used to provide a standard interface for passing parameters to the program. The first six words of the TWA are reserved for use by ADABAS SQL to communicate with CICS. This contains the addresses of the ADABAS Control Block, Format Buffer, Record Buffer, Search Buffer, Value Buffer, and ISN Buffer.

A Cobol or Pl/1 program should specify the following option:

```
ADACALL ADASTWA USING TWA.  
TELE "EXEC CICS LINK PROGRAM  
      ('ADABAS') END-EXEC".
```

This will cause each call to ADABAS to be substituted with a call to ADASTWA.

ADABAS SQL Coding Considerations

- Calls may be imbedded in either COBOL, PL/1, FORTRAN, or Ada programs.
- Comments may not be imbedded within the ADABAS SQL commands.
- Host language source code may not be imbedded within the ADABAS SQL command.
- ADABAS SQL commands may not be imbedded within a COBOL IF-THEN-ELSE clause.
- The EXEC ADABAS command must be on one line.
- The END-EXEC command must be on one line.
- Only one SQL command may be between the EXEC ADABAS command and the END-EXEC.
- The ADABAS SQL command may contain up to 20 lines.
- Variable indexes should be avoided wherever possible since they tend to cause additional commands to be executed (Format Translations and RC Commands)
- The number of fields that may be selected cannot total greater than 50 in any one SELECT.
- Instead of PERFORMing a section of code (series of statements) repeatedly, code and use the Command-ID by way of the cursor defined in the DECLARE option; reduces the number of unnecessary translations

Global Parameters

For a detailed discussion of each parameter, see Software AG's *ADABAS SQL Reference Manual*, Manual Order Number: SQL-140-030.

ABORT: used to modify ADABAS SQL's action when an ADABAS response code other than "0" or "3" is encountered.

ADACALL: used to generate non-standard ADABAS calls.

APOS: determines if character strings generated by ADABAS SQL should be enclosed in single or double quotes.

DDFILE: provides ADABAS SQL with the file number of the PREDICT data dictionary file.

LANG: determines if data declarations and code should be generated for Ada, Cobol, COBOL/II, Fortran, Fortran/VMS, or PL/1.

MODE: controls the debugging facilities which are built into ADABAS SQL.

MONITOR: eliminates need for coding ADACALL and TELE statements in COBOL programs necessary when implementing programs under CICS.

NAME: indicates the name of the program as it is defined to the PREDICT data dictionary.

OPTIONS: specifies whether or not dynamic Command IDs are to be generated; specifies the length of the Buffer which contains ET data; specifies the size of the ISN Buffer, etc.

SYSFILE: points to PREDICT and NATURAL System Files.

TELE: specifies a source statement which is to be inserted after each generated call statement.

USER: identifies the user creating program, optionally stored on PREDICT if XREF is on. ADABAS SQL uses the first three characters of the program name as the default userID.

XREF: controls the creation of cross-reference data in PREDICT data dictionary.

ADABAS/ADAMINT/NATURAL Equivalent Commands

ADABAS SQL	ADABAS	ADAMINT	NATURAL
CONNECT	OP	SIGNON MULTOPEN	nothing in NATURAL
DBCLOSE	CL	SIGNOFF MULTCLOS	nothing in NATURAL
FIND & OPEN	S1 / S4	FINDSET	FIND (NUMBER)
FETCH	L1 / L4	READSET	FIND
READ LOGICAL & OPEN	---	LOKATE	nothing in NATURAL
FETCH	L3 / L6	SEQREAD	READ LOGICAL
HISTOGRAM & OPEN	---	LOKVAL	nothing in NATURAL
FETCH	L9	READVAL	HISTOGRAM
READ PHYSICAL & OPEN	---	LOKATE	nothing in NATURAL
FETCH	L1 / L5	SEQREAD	READ PHYSICAL
READ ISN & OPEN	---	LOKATE	nothing in NATURAL
FETCH	L1 / L4	SEQREAD	READ ISN
use HOLD for REREAD with HOLD	L4	REREAD	GET
HOLD	HI	---	nothing in NATURAL
INSERT	NI	ADDNEW	STORE
UPDATE	A1	UPDATER	UPDATE
DELETE	E1	DELETER	DELETE
RELEASE ISN	R1	RELEASER	nothing in NATURAL
COMMIT	ET	MINTET	END TRANSACTION
ROLLBACK	BT	MINTBT	nothing in NATURAL
COMPARE ISN & OPEN & FETCH	S8	COMPSET	nothing in NATURAL
SORT ISN & OPEN & FETCH	S9	SORTSET	FIND SORTED BY
READ USERDATA	RE	MINTRE	READ DATA
CHECKPOINT	C1/C2/C3	CHEKPNT	nothing in NATURAL
	C1	MINTC1	nothing in NATURAL
	C2	MINTC2	
	C3	MINTC3	

ADABAS/ADAMINT/NATURAL Equivalent Commands

ADABAS SQL	ADABAS	ADAMINT	NATURAL
WRITE TO LOG	C5	LOGTAPE	nothing in NATURAL
	C5	MINTC5	nothing in NATURAL
RELEASE	RC	RELCID	nothing in NATURAL
TRACE ON	---	SNAPINT MULTSNAP	nothing in NATURAL
use MOVE ISN IN	---	RETISN	nothing in NATURAL

Shell Examples

Data Retrieval

```

EXEC ADABAS
    BEGIN DECLARE SECTION
END-EXEC
.
.
EXEC ADABAS
    DECLARE PERS CURSOR FOR
    SELECT LAST-NAME,
           FIRST-NAME,
           MIDDLE-INIT,
           SEX,
           SALARY
    FROM PERSONNEL
    WHERE LAST-NAME= 'HARRIS '
END-EXEC
.
.
EXEC ADABAS
    OPEN PERS
END-EXEC
.
.
EXEC ADABAS
    FETCH PERS
END EXEC
.
.
PERFORM READ-PERS
    UNTIL ADACODE=3.
.
.
EXEC ADABAS
    CLOSE PERS
END-EXEC
STOP RUN.
.
.

```

```

READ-PERS.
  DISPLAY LAST-NAME FIRST-NAME
          MIDDLE-INIT AGE SEX
          SALARY.
EXEC ADABAS
  FETCH PERS
END-EXEC

```

Data Creation

```

EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
.
EXEC ADABAS
  INSERT INTO PERSONNEL
  SET LAST-NAME='HARRIS'
  FIRST-NAME='CODY'
  MIDDLE-INIT='A'
  SEX='M'
  SALARY=146500
END-EXEC
.
EXEC ADABAS
  COMMIT WORK
END-EXEC

```

Data Modification

```

EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
.
EXEC ADABAS
  DECLARE PERS CURSOR
  SELECT LAST-NAME,
         FIRST-NAME,
         MIDDLE-INIT,
         SEX,
         SALARY
  FROM PERSONNEL
  WHERE LAST-NAME='HARRIS'
  OPTIONS HOLD
END-EXEC
.
EXEC ADABAS
  UPDATE PERSONNEL
  SET SALARY=54500
  WHERE CURRENT OF PERS
END-EXEC
.
EXEC ADABAS
  COMMIT WORK
END-EXEC
.

```

```

.
EXEC ADABAS
      DBCLOSE
END-EXEC

```

Data Deletion

```

EXEC ADABAS
      BEGIN DECLARE SECTION
END-EXEC
.
EXEC ADABAS
      DECLARE PERS CURSOR FOR
      FROM PERSONNEL
      WHERE PERSONNEL-NUMBER=070187
      OPTIONS HOLD
END-EXEC
.
EXEC ADABAS
      FETCH PERS
END-EXEC
.
EXEC ADABAS
      DELETE
      WHERE CURRENT OF PERS
END-EXEC
.
EXEC ADABAS
      COMMIT WORK
END-EXEC

```

- 1 In this simple COBOL example, several fields are to be read from the VEHICLES file for all the Porsche automobiles.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEADER-LINE-1.
   02 FILLER PIC X(10) VALUE '.....MA'.
   02 FILLER PIC X(10) VALUE 'KE.....'.
   02 FILLER PIC X(10) VALUE '.....MO'.
   02 FILLER PIC X(10) VALUE 'DEL.....'.
   02 FILLER PIC X(10) VALUE '....COLOR.'.
   02 FILLER PIC X(10) VALUE '...YEAR.RE'.
   02 FILLER PIC X(10) VALUE 'GISTRATION'.
EXEC ADABAS
      BEGIN DECLARE SECTION
END-EXEC
EXEC ADABAS
      DECLARE CARS CURSOR FOR
      SELECT MAKE, MODEL, COLOR, YEAR,

```

```

      REG-NUM
      FROM VEHICLES
      WHERE MAKE = 'PORSCHÉ'
      END-EXEC
PROCEDURE DIVISION.
* * * * *
* OPEN ADABAS FILE *
* * * * *
      EXEC ADABAS
      OPEN CARS
      END-EXEC
      DISPLAY HEADER-LINE-1.
* * * * *
* READ (FIRST) ADABAS DATA (RECORD) *
* * * * *
      EXEC ADABAS
      FETCH CARS
      END-EXEC
      PERFORM READ-CARS UNTIL ADACODE = 3.
* * * * *
* CLOSE ADABAS FILE *
* * * * *
      EXEC ADABAS
      CLOSE CARS
      END-EXEC
      STOP RUN.
READ-CARS.
      DISPLAY MAKE MODEL COLOR YEAR REG-NUM.
* * * * *
* READ MORE ADABAS DATA *
* * * * *
      EXEC ADABAS
      FETCH CARS
      END-EXEC

```

COBOL code generated by ADABAS SQL for MRSTMEX1.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEADER-LINE-1.
   02 FILLER PIC X(10) VALUE '.....MA'.
   02 FILLER PIC X(10) VALUE 'KE.....'.
   02 FILLER PIC X(10) VALUE '.....MO'.
   02 FILLER PIC X(10) VALUE 'DEL.....'.
   02 FILLER PIC X(10) VALUE '....COLOR.'.
   02 FILLER PIC X(10) VALUE '...YEAR.RE'.
   02 FILLER PIC X(10) VALUE 'GISTRATION'.
*
* EXEC ADABAS
*   BEGIN DECLARE SECTION
* END-EXEC
*
01 ADACODE PIC 9(4) COMP VALUE 0.
01 CONTROL-BLOCKOPN.
   03 FILLER1OPN PIC XX VALUE 'AS'.
   03 COMMAND-CODEOPN PIC XX VALUE SPACE.

```

```

03 COMMAND-IDOPN          PIC X(4)      VALUE 'OPEN' .
03 FILE-NUMBEROPN        PIC 9(4)     COMP VALUE 0 .
03 FILLER REDEFINES FILE-NUMBEROPN.
  04 DBIDOPN              PIC X .
  04 FILLER                PIC X .
03 RESPONSE-CODEOPN      PIC 9(4)     COMP VALUE 0 .
03 ISNOPN                 PIC 9(9)     COMP VALUE 0 .
03 ISN-LOWER-LIMIT       PIC 9(9)     COMP VALUE 0 .
03 ISN-QUANTITYOPN       PIC 9(9)     COMP VALUE 0 .
03 FORMAT-BUFFER-LENGTHOPN PIC 9(4)     COMP VALUE 0 .
03 RECORD-BUFFER-LENGTHOPN PIC 9(4)     COMP VALUE 0 .
03 SEARCH-BUFFER-LENGTHOPN PIC 9(4)     COMP VALUE 0 .
03 VALUE-BUFFER-LENGTHOPN PIC 9(4)     COMP VALUE 0 .
03 ISN-BUFFER-LENGTHOPN  PIC 9(4)     COMP VALUE 0 .
03 COMMAND-OPTION-1OPN   PIC X        VALUE SPACE .
03 COMMAND-OPTION-2OPN   PIC X        VALUE SPACE .
03 ADDITIONS-1OPN        VALUE SPACE .
  04 ADDITIONS-1-12OPN    PIC XX .
  04 FILLER                PIC XX .
  04 ADDITIONS-1-58OPN    PIC X(4) .
03 FILLER REDEFINES ADDITIONS-1OPN.
  04 ADDITIONS-1-BNOPN    PIC 9(4)     COMP .
  04 FILLER PIC X(6) .
03 ADDITIONS-2OPN        PIC X(4)     VALUE SPACE .
03 ADDITIONS-3OPN        PIC X(4)     VALUE SPACE .
03 ADDITIONS-4OPN        PIC X(4)     VALUE SPACE .
03 ADDITIONS-5OPN .
  04 ADDITIONS-5-BNOPN    PIC 9(9)     COMP VALUE 0 .
  04 ADDITIONS-5-58OPN    PIC X(4)     VALUE SPACE .
03 FILLER REDEFINES ADDITIONS-5OPN.
  04 ADDITIONS-5-1OPN     PIC X .
  04 ADDITIONS-5-28OPN    PIC X(7) .
03 COMMAND-TIMEOPN       PIC 9(9)     COMP .
03 USER-AREAOPN          PIC X(4)     VALUE 'AS' .
01 FORMAT-BUFOPN         PIC X .
01 SEARCH-BUFOPN         PIC X .
01 VALUE-BUFOPN          PIC X .
01 RECORD-BUFOPN         PIC X(1500) .
01 OPENTYPE              PIC X(0010) .
01 DDFILE                 PIC 999     VALUE 17 .
01 CSEQ                   PIC X(8) .
01 CLN1 .
  02 CLN1V                 PIC X(40)   OCCURS 20 .
01 CLN2 .
  02 CLN2V                 PIC X(40)   OCCURS 20 .
01 CLNNUM                 PIC 9(4)     COMP .
01 TRCE                   PIC X(7) .
01 SAVE-DBID-1OPN        PIC 9(4)     COMP .
01 SAVE-DBID-DEFOPN REDEFINES SAVE-DBID-1OPN.
  02 FILLER                PIC X .
  02 SAVE-DBIDOPN         PIC X .
01 FORMAT-BUFCARS .
  02 FILLER                PIC X(01)   VALUE ' ' .
01 SEARCH-BUFCARS .
  02 FILLER                PIC X(31)   VALUE 'AA,020,A.'
01 HUGO .
  02 RECORD-BUFCARS       PIC X .
  02 ISN                   PIC 9(9)     COMP .
  02 QUANTITY              PIC 9(9)     COMP .
  02 RESPONSE-CODE        PIC 9(4)     COMP .

```

```

01 VALUE-BUFCARS.
  02 V-MAKE-01          PIC X(20) VALUE 'PORSCHÉ'.
01 ISN-BUFCARS.
  02 ISN-BUFVECCARS OCCURS 1 PIC 9(9) COMP.
01 CONTROL-BLOCKCARS.
  03 FILLER1CARS        PIC XX          VALUE 'AS'.
  03 COMMAND-CODECARS   PIC XX          VALUE SPACE.
  03 COMMAND-IDCARS     PIC X(4)        VALUE 'CARS'.
  03 FILE-NUMBERCARS    PIC 9(4) COMP VALUE 0.
  03 FILLER REDEFINES FILE-NUMBEROPN.
    04 DBIDCARS         PIC X.
    04 FILLER           PIC X.
  03 RESPONSE-CODECARS PIC 9(4) COMP VALUE 0.
  03 ISNOPN             PIC 9(9) COMP VALUE 0.
  03 ISN-LOWER-LIMIT   PIC 9(9) COMP VALUE 0.
  03 ISN-QUANTITYOPN   PIC 9(9) COMP VALUE 0.
  03 FORMAT-BUFFER-LENGTHCARS PIC 9(4) COMP VALUE 1.
  03 RECORD-BUFFER-LENGTHCARS PIC 9(4) COMP VALUE 1.
  03 SEARCH-BUFFER-LENGTHCARS PIC 9(4) COMP VALUE 12.
  03 VALUE-BUFFER-LENGTHCARS PIC 9(4) COMP VALUE 7.
  03 ISN-BUFFER-LENGTHCARS PIC 9(4) COMP VALUE 4.
  03 COMMAND-OPTION-1CARS PIC X          VALUE SPACE.
  03 COMMAND-OPTION-2CARS PIC X          VALUE SPACE.
  03 ADDITIONS-1CARS    VALUE SPACE.
    04 ADDITIONS-1-12CARS PIC XX.
    04 FILLER           PIC XX.
    04 ADDITIONS-1-58CARS PIC X(4).
  03 FILLER REDEFINES ADDITIONS-1CARS.
    04 ADDITIONS-1-BNCARS PIC 9(4) COMP.
    04 FILLER PIC X(6).
  03 ADDITIONS-2CARS    PIC X(4)        VALUE SPACE.
  03 ADDITIONS-3CARS    PIC X(4)        VALUE SPACE.
  03 ADDITIONS-4CARS    PIC X(4)        VALUE SPACE.
  03 ADDITIONS-5CARS.
    04 ADDITIONS-5-BNCARS PIC 9(9) COMP VALUE 0.
    04 ADDITIONS-5-58CARS PIC X(4)        VALUE SPACE.
  03 FILLER REDEFINES ADDITIONS-5CARS.
    04 ADDITIONS-5-1CARS  PIC X.
    04 ADDITIONS-5-28CARS PIC X(7).
  03 COMMAND-TIMECARS   PIC 9(9) COMP.
  03 USER-AREACARS     PIC X(4)        VALUE 'AS'.
01 ISNSIZECARS         PIC 9(9) COMP.
01 ISNMORECARS         PIC 9(9) COMP.
01 ISNINDCARS          PIC 9(4) COMP.
01 EOF-COBCARS         PIC 9          VALUE 0.
  88 EOF-CARS VALUE 1.
  88 NOT-EOF-CARS VALUE 0.
01 SAVE-DBID-1CARS     PIC 9(4) COMP.
01 SAVE-DBID-DEFCARS REDEFINES SAVE-DBID-1CARS.
  02 FILLER            PIC X.
  02 SAVE-DBIDCARS     PIC X.
01 FORMAT-BUFCARS.
  02 FILLER            PIC X(45) VALUE
    'AA,020,A,AB,020,A,AC,010,A,AD,002,U,AE,010,A.'.
01 SEARCH-BUFCARS.
  02 FILLER            PIC X(01) VALUE ' '.
01 CARS.
  02 MAKE              PIC X(20).
  02 MODEL             PIC X(20).

```

```

02 COLOR PIC X(10).
02 YEAR PIC 9(2).
02 REG-NUM PIC X(10).
01 VALUE-BUFCARS PIC X.
01 ISN-BUFCARS.
02 ISN-BUFVECCARS OCCURS 1 PIC 9(9) COMP.
EXEC ADABAS
    DECLARE CARS CURSOR FOR
        SELECT MAKE, MODEL, COLOR, YEAR,
            REG-NUM
        FROM VEHICLES
        WHERE MAKE = 'PORSCHE'
    END-EXEC
PROCEDURE DIVISION.
* EXEC ADABAS
* OPEN CARS
* END-EXEC
DISPLAY HEADER-LINE-1.
* EXEC ADABAS
* FETCH CARS
* END-EXEC
PERFORM READ-CARS UNTIL ADACODE = 3.
* EXEC ADABAS
* CLOSE CARS
* END-EXEC
STOP RUN.
READ-CARS.
DISPLAY MAKE MODEL COLOR YEAR REG-NUM.
* EXEC ADABAS
* FETCH CARS
* END-EXEC

```

- 2 In this simple COBOL example, several fields are to be read from the EMPLOYEES file for all the residents of Paris.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEADER-LINE-1.
02 FILLER PIC X(10) VALUE 'PERSONNEL.'.
02 FILLER PIC X(10) VALUE '.....NAM'.
02 FILLER PIC X(10) VALUE 'E.....'.
02 FILLER PIC X(10) VALUE '..FIRST-NA'.
02 FILLER PIC X(10) VALUE 'ME..M..ANN'.
02 FILLER PIC X(10) VALUE 'UAL.....'.
02 FILLER PIC X(10) VALUE '.....'.
02 FILLER PIC X(10) VALUE '..JOB TITL'.
02 FILLER PIC X(10) VALUE 'E.....'.
02 FILLER PIC X(10) VALUE 'DEPT.....'.
01 HEADER-LINE-2.
02 FILLER PIC X(10) VALUE '.NUMBER...'.
02 FILLER PIC X(30) VALUE SPACES.
02 FILLER PIC X(10) VALUE '....I..SAL'.
02 FILLER PIC X(10) VALUE 'ARY....BON'.
02 FILLER PIC X(10) VALUE 'US.....'.
02 FILLER PIC X(20) VALUE SPACES.
02 FILLER PIC X(10) VALUE 'CODE.....'.
EXEC ADABAS
BEGIN DECLARE SECTION
END-EXEC

```

```

EXEC ADABAS
  DECLARE EMPS CURSOR FOR
  SELECT PERSONNEL-ID, NAME,
         FIRST-NAME, MIDDLE-NAME
         SALARY(1), BONUS(1(1)), <--- watch this type of code
         JOB-TITLE, DEPT
  FROM EMPLOYEES
  WHERE CITY = 'PARIS' <--- or as a variable CITY = :var
END-EXEC
PROCEDURE DIVISION.
* * * * *
* OPEN ADABAS FILE *
* * * * *
  EXEC ADABAS
    OPEN EMPS
  END-EXEC
  DISPLAY HEADER-LINE-1.
  DISPLAY HEADER-LINE-2.
* * * * *
* READ (FIRST) ADABAS DATA (RECORD) *
* * * * *
  EXEC ADABAS
    FETCH EMPS
  END-EXEC
  PERFORM READ-EMPS UNTIL ADACODE = 3. <---response code 3 = end of file
* * * * *
* CLOSE ADABAS FILE *
* * * * *
  EXEC ADABAS
    CLOSE EMPS
  END-EXEC
  STOP RUN.
READ-EMPS.
  DISPLAY PERSONNEL-ID LAST-NAME
         FIRST-NAME MIDDLE-NAME
         SALARY(1) BONUS(1,1)
         JOB-TITLE DEPT.
* * * * *
* READ MORE ADABAS DATA *
* * * * *
  EXEC ADABAS
    FETCH EMPS
  END-EXEC

```

- ③ In this COBOL example, all people on the EMPLOYEES file who work in the computer department: DEPT equals a value COMP00 thru COMP99. For each employee, name job-title and department is listed.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX3.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEADER-LINE-1.
   02 FILLER PIC X(10) VALUE '.....NAM'.
   02 FILLER PIC X(10) VALUE 'E.....'.

```

```

02 FILLER PIC X(10) VALUE '..FIRST-NA'.
02 FILLER PIC X(10) VALUE 'ME.....DEP'.
02 FILLER PIC X(10) VALUE 'T.....'.
02 FILLER PIC X(10) VALUE 'JOB TITLE.'.
01 HEADER-LINE-2.
02 FILLER PIC X(60) VALUE '-'.
EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
EXEC ADABAS
  READ LOGICAL
  DECLARE EMPS CURSOR FOR
  SELECT NAME, FIRST-NAME,
         JOB-TITLE, DEPT
  FROM EMPLOYEES
  WHERE DEPT GE 'COMP00'
END-EXEC
PROCEDURE DIVISION.
* * * * *
* OPEN ADABAS FILE *
* * * * *
EXEC ADABAS
  OPEN EMPS
END-EXEC
  DISPLAY HEADER-LINE-1.
  DISPLAY HEADER-LINE-2.
* * * * *
* READ (FIRST) ADABAS DATA (RECORD) *
* * * * *
EXEC ADABAS
  FETCH EMPS
END-EXEC
  PERFORM READ-EMPS UNTIL ADACODE = 3
                                OR DEPT = 'COMP99'.
* * * * *
* CLOSE ADABAS FILE *
* * * * *
EXEC ADABAS
  CLOSE EMPS
END-EXEC
  STOP RUN.
READ-EMPS.
  DISPLAY LAST-NAME FIRST-NAME
         JOB-TITLE DEPT.
* * * * *
* READ MORE ADABAS DATA *
* * * * *
EXEC ADABAS
  FETCH EMPS
END-EXEC

```

- [4] In this COBOL example, automobiles of each person on the EMPLOYEES file who live in Paris are listed. For each employee, name job-title and make of car are listed. (PERSONNEL-ID is used to logically link the files).

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX4.
ENVIRONMENT DIVISION.

```

```

INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HEADER-LINE-0.
    02 FILLER PIC X(20) VALUE SPACE.
    02 FILLER PIC X(15) VALUE 'PARIS RESIDENTS'.
    02 FILLER PIC X(25) VALUE SPACE.
01 HEADER-LINE-1.
    02 FILLER PIC X(10) VALUE '.....NAM'.
    02 FILLER PIC X(10) VALUE 'E.....'.
    02 FILLER PIC X(10) VALUE SPACE.
    02 FILLER PIC X(10) VALUE 'JOB TITLE.'.
    02 FILLER PIC X(10) VALUE SPACE.
    02 FILLER PIC X(10) VALUE '..MAKE....'.
01 HEADER-LINE-2.
    02 FILLER PIC X(60) VALUE '-'.
EXEC ADABAS
    BEGIN DECLARE SECTION
END-EXEC
EXEC ADABAS
    FIND
    DECLARE EMPS CURSOR FOR
    SELECT NAME, JOB-TITLE, PERSONNEL-ID
    FROM EMPLOYEES
    WHERE CITY = 'PARIS'
END-EXEC
EXEC ADABAS
    FIND
    DECLARE CARS CURSOR FOR
    SELECT MAKE
    FROM VEHICLES
    WHERE PERSONNEL-ID = EMPS.PERSONNEL-ID
END-EXEC
PROCEDURE DIVISION.
* * * * *
* LINK EMPLOYEES AND VEHICLES *
* * * * *
EXEC ADABAS
    CONNECT ACC=EMPLOYEES VEHICLES
END-EXEC
* * * * *
* OPEN ADABAS FILE *
* * * * *
EXEC ADABAS
    OPEN EMPS
END-EXEC
    DISPLAY HEADER-LINE-0.
    DISPLAY HEADER-LINE-1.
    DISPLAY HEADER-LINE-2.
* * * * *
* READ (FIRST) ADABAS DATA (RECORD) *
* * * * *
EXEC ADABAS
    FETCH EMPS
END-EXEC
    PERFORM READ-EMPS UNTIL ADACODE = 3.
* * * * *
* CLOSE ADABAS FILE *
* * * * *

```

```

EXEC ADABAS
  CLOSE EMPS
END-EXEC
* * * * *
* END ADABAS SESSION *
* * * * *
EXEC ADABAS
  DBCLOSE
END-EXEC
STOP RUN.
READ-EMPS.
* * * * *
* FIND VEHICLE DATA *
* * * * *
EXEC ADABAS
  OPEN CARS
END-EXEC
EXEC ADABAS
  FETCH CARS
END-EXEC
PERFORM READ-CARS UNTIL ADACODE = 3.
EXEC ADABAS
  CLOSE CARS
END-EXEC
* * * * *
* READ MORE ADABAS DATA *
* * * * *
EXEC ADABAS
  FETCH EMPS
END-EXEC
READ-CARS.
EXEC ADABAS
  FETCH CARS
END-EXEC
DISPLAY LAST-NAME JOB-TITLE MAKE.

```

- 5] Several fields are to be read from the VEHICLES file for all the Porsche automobiles. Conversion of colors to English for all records is executed in this COBOL example.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MRSTMEX5.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NEW-COLOR PIC X(10).
EXEC ADABAS
  BEGIN DECLARE SECTION
END-EXEC
EXEC ADABAS
  DECLARE CARS CURSOR FOR
  SELECT MAKE, COLOR
  FROM VEHICLES
  WHERE MAKE = 'PORSCHE'
  OPTIONS HOLD
END-EXEC

```

```

PROCEDURE DIVISION.
* * * * *
* OPEN ADABAS FILE *
* * * * *
    EXEC ADABAS
      OPEN CARS
    END-EXEC
* * * * *
* READ (FIRST) ADABAS DATA (RECORD) *
* * * * *
    EXEC ADABAS
      FETCH CARS
    END-EXEC
    PERFORM READ-AND-UPDATE-CARS UNTIL ADACODE = 3.
* * * * *
* POST CHANGE TO ADABAS FILE *
* * * * *
    EXEC ADABAS
      COMMIT WORK
    END-EXEC
* * * * *
* CLOSE ADABAS FILE *
* * * * *
    EXEC ADABAS
      CLOSE CARS
    END-EXEC
    STOP RUN.
READ-AND-UPDATE-CARS.
  IF COLOR OF CARS IS 'ROUGE' OR 'ROT'
    THEN MOVE 'RED' TO NEW-COLOR
  ELSE IF COLOR OF CARS IS 'BLAU' OR 'AZUL'
    THEN MOVE 'BLUE' TO NEW-COLOR
  ELSE IF COLOR OF CARS IS 'VERT' OR 'VERDE'
    THEN MOVE 'GREEN' TO NEW-COLOR
  ELSE IF COLOR OF CARS IS 'BLANCO' OR 'BLANC'
    THEN MOVE 'WHITE' TO NEW-COLOR
  ELSE IF COLOR OF CARS IS 'NOIR' OR 'NEGRO'
    THEN MOVE 'BLACK' TO NEW-COLOR
  ELSE MOVE 'NO COLOR' TO NEW-COLOR.
* * * * *
* MODIFY ADABAS DATA *
* * * * *
    EXEC ADABAS
      UPDATE VEHICLES
        SET COLOR = :NEW-COLOR
        WHERE CURRENT OF CARS
    END-EXEC
* * * * *
* READ MORE ADABAS DATA *
* * * * *
    EXEC ADABAS
      FETCH CARS
    END-EXEC

```

