

Adabas Native SQL

Adabas Native SQL Reference Manual

Version 2.4.1

June 2014

This document applies to Adabas Native SQL Version 2.4.1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Document ID: SQL-REFERENCE-241-20140626

Table of Contents

Adabas Native SQL Reference Manual	vii
1 INTRODUCTION	1
2 PROGRAMMING CONSIDERATIONS	5
Rules for Adabas Native SQL Statements	7
Source Program Maintenance	8
The Record Buffer and Reference to Data	9
Response Code Interpretation	26
Host Variables	28
ISN Lists and the ISN Buffer	28
HOLD Logic	29
Security Options	30
Record Buffer - ADA	31
Record Buffer - COBOL	31
Fields in FORTRAN	32
Record Buffer - PL/I	33
Date and Time Conversion Routines	34
Support of Distributed Data Structures	35
The Distribution handling	36
Relational Null Support	37
Long Alpha field Support	38
3 SINGLE AND MULTIPLE-RECORD PROCESSING	39
Single-Record Processing	40
Multiple-Record Processing	41
4 OVERVIEW OF STATEMENTS	43
Syntax	44
Overview of Adabas Native SQL Statements	47
Database Query Statements	47
Data Storage READ Statements	48
Associator READ Statement	49
Statements for Processing Multiple Records	50
Database Modification Statements	50
Logical Transaction Processing Statements	51
Checkpointing Statement	52
Other Adabas Native SQL Statements	52
Adabas Native SQL Clauses	53
5 ADABAS NATIVE SQL STATEMENTS	71
The BEGIN Statement	72
The CHECKPOINT Statement	73
The CLOSE Statement	74
The COMMIT WORK Statement	75
The COMPARE Statement	77
The CONNECT Statement	87
The COPY Statement	97

The DBCLOSE Statement	98
The DELETE Statement	101
The FETCH Statement	106
The FIND Statement	107
The FIND COUPLED Statement	125
The GENERATE Statement	135
The HISTOGRAM Statement	136
The HOLD Statement	146
The INSERT Statement	148
The OPEN Statement	157
The READ ISN Statement	158
The READ LOGICAL Statement	167
The READ PHYSICAL SEQUENCE Statement	179
The READ USERDATA Statement	187
The RELEASE Statement	189
The RELEASE ISN Statement	190
The RESTORE Statement	191
The ROLLBACK WORK Statement	192
The SAVE Statement	194
The SORT Statement	196
The TRACE Statement	207
The UPDATE Statement	208
The WHENEVER Statement	220
The WRITE TO LOG Statement	318
6 USING ADABAS NATIVE SQL STATEMENTS IN TP PROGRAMS	223
COM-LETE	224
Customer Information Control System (CICS)	224
7 GLOBAL PARAMETERS	227
The ABORT Parameter	229
The ADACALL Parameter	232
The APOS Parameter	234
The CICS STUB Parameter	235
The LANG Parameter	236
The LIBRARY Parameter	237
The MODE Parameter	238
The MONITOR Parameter	240
The NAME Parameter	243
The NETWORK Parameter	244
The OPTIONS Parameter	245
The SYSDATA Parameter	256
The TELE Parameter	257
The USER Parameter	258
The VIRTUAL-MACHINE Parameter	259
The XREF Parameter	260
8 APPENDIX A - SIZE LIMITATIONS	261

9 APPENDIX B - DESCRIPTIONS OF THE FILES USED IN THE EXAMPLES	263
FORTRAN Synonyms	265
10 APPENDIX C - ADABAS NATIVE SQL STATEMENTS USED IN THE EXAMPLES	267
11 APPENDIX D - ADA EXAMPLES	269
Example 1	270
Example 2	271
Example 3	274
12 APPENDIX E - EXAMPLE OF ADA CODE GENERATED BY ADABAS NATIVE SQL	279
13 APPENDIX F - COBOL EXAMPLES	287
Example 1	288
Example 2	290
Example 3	292
14 APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL	297
15 APPENDIX H - FORTRAN EXAMPLES	315
Example 1	316
Example 2	317
Example 3	320
16 APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL	325
17 APPENDIX J - PL/I EXAMPLES	335
Example 1	336
Example 2	338
Example 3	340
18 APPENDIX - EXAMPLE OF PL/I CODE GENERATED BY ADABAS NATIVE SQL	345

Adabas Native SQL Reference Manual

This document describes the functions provided by Adabas Native SQL, Software AG's language for accessing Adabas files from Ada, COBOL, FORTRAN and PL/I programs. SQL stands for Structured Query Language. This document also describes how to code the statements that provide these functions.

The document's intended audience is an Ada, COBOL, FORTRAN77 or PL/I programmer who is also acquainted with Adabas concepts and who wishes to develop applications using Adabas Native SQL.

This documentation consists of the following sections:

Introduction	This describes the basic concepts of Adabas Native SQL.
Programming Considerations	This provides background information you should read before using Adabas Native SQL for the first time. This material will help you understand <ul style="list-style-type: none">■ the data structures that Adabas Native SQL builds in your programs,■ how your program should react if Adabas Native SQL detects an error,■ how Adabas Native SQL reads lists of records in sequence,■ how to hold records in order to avoid updating conflicts, and■ how to access and update files that are protected by the Adabas security mechanisms.
Single and Multiple-Record Processing	This also includes a section on distributed data processing. This deals with considerations when operating in single or multiple-record processing mode.
Overview of Statements	This provides an overview of the syntax used in Adabas Native SQL statements, together with a brief description of the statements themselves, grouped logically according to statement function. This chapter also describes in detail the clauses common to statements which retrieve data from the database.
Adabas Native SQL Statements	This describes in detail all the statements in alphabetical order for easy reference.
Using Adabas Native SQL Statements in TP Programs	This provides additional information on the facilities provided for writing teleprocessing (TP) application programs.
Global Parameters	This describes global parameters which can be used to define processing options and adapt them to your particular requirements.
Appendix A: Size Limitations	Lists the size limitations of Adabas Native SQL.
Appendix B: Descriptions of the Files used in the Examples	Contains a description of the files used in the sample programs and the FORTRAN synonyms that must be used.

Appendix C	Adabas Native SQL statements used in the examples.
Appendix D	ADA Examples
Appendix E	Example of ADA code generated by Adabas Native SQL.
Appendix F	COBOL Examples
Appendix G	Example of COBOL code generated by Adabas Native SQL.
Appendix H	FORTRAN Examples
Appendix I	Example of FORTRAN code generated by Adabas Native SQL.
Appendix J	PL/I Examples
Appendix K	Example of PL/I code generated by Adabas Native SQL.

Other Sources of Information

This reference guide, read in conjunction with the Adabas Introduction Manual, should provide all the information that you need when writing Adabas Native SQL application programs. However, when writing TP application programs or if the database is protected by the Adabas security features, you may need to refer to other sources, for example the database administrator (DBA) or the following literature:

- *Adabas Operations Manual*
- *Adabas Utilities Manual*
- *Adabas DBA Reference Manual*
- *Adabas Command Reference Manual*
- *Adabas Installation Manual*
- *Adabas Messages and Codes.*

1 INTRODUCTION

Adabas Native SQL is an easy-to-use data manipulation language for accessing and updating information held in an Adabas database. The following example shows a typical Adabas Native SQL statement that selects a record from the database and retrieves the required data:

```
EXEC ADABAS
  SELECT NAME, AGE, SALARY
  FROM PERSONNEL
  WHERE NUMBER-OF-DEPENDENTS > 4
END-EXEC
```

This statement selects the data fields *NAME*, *AGE* and *SALARY* from the first record in the *PERSONNEL* file that satisfies the criterion "NUMBER-OF-DEPENDENTS > 4".

Statements such as this one are embedded into Ada, COBOL, FORTRAN77 or PL/I programs. This means you have the advantage of being able to use a familiar programming language to code the logic of your problem, whilst the Adabas Native SQL statements give you ready access to all the facilities of Adabas, a powerful modern database management system.

Adabas Native SQL incorporates the full power of the Natural userview concept. This means you refer to fields defined in a userview as logical entities without having to concern yourself with the physical details of file structure and record structure. For example, if you specify a group field, Adabas Native SQL automatically creates Ada, COBOL, FORTRAN or PL/I data declarations with the correct:

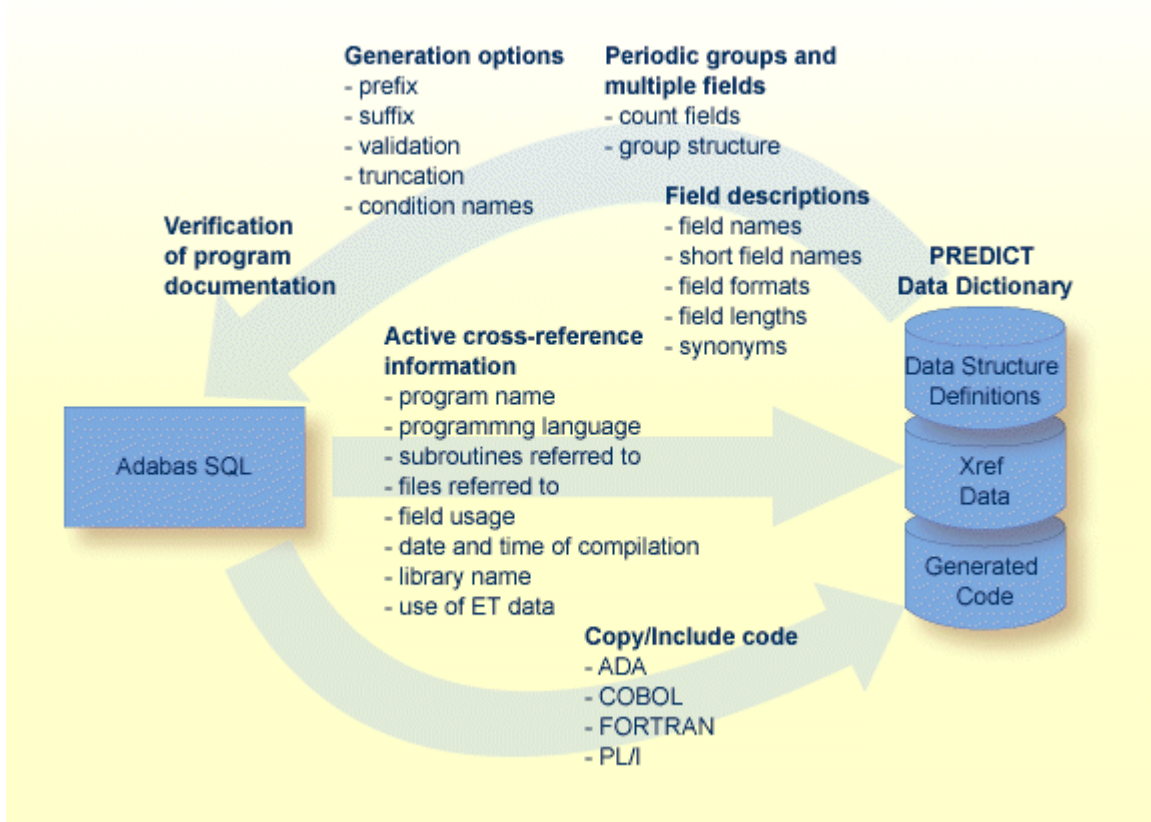
- set of fields (possibly a subset of the fields in the database record; conversely, a field may occur repeatedly in the userview if desired)
- field names
- field sequence
- record structure, including all groups, sub-groups, sub-sub-groups, etc.

- field formats (alphanumeric, numeric, packed numeric, etc.)
- field lengths.

Adabas Native SQL works in conjunction with Predict, Software AG's data dictionary system. The information about file and record layouts contained in Predict is used to generate the data structures that the generated Ada, COBOL, FORTRAN or PL/I program needs to access the database. As an Adabas Native SQL programmer, you do not need to code detailed data declarations in your program, so you are free to concentrate on the logic of the application.

Conversely, as Adabas Native SQL is processing the program, it records active cross-reference information, or Xref data, in Predict. This Xref data includes the names of the files and fields that the program accesses. Thus it is easy to find out which programs use which data fields, etc., so that the programs that need to be recompiled when data structures are altered can readily be determined.

The interaction between Adabas Native SQL and Predict is illustrated in the following figure.



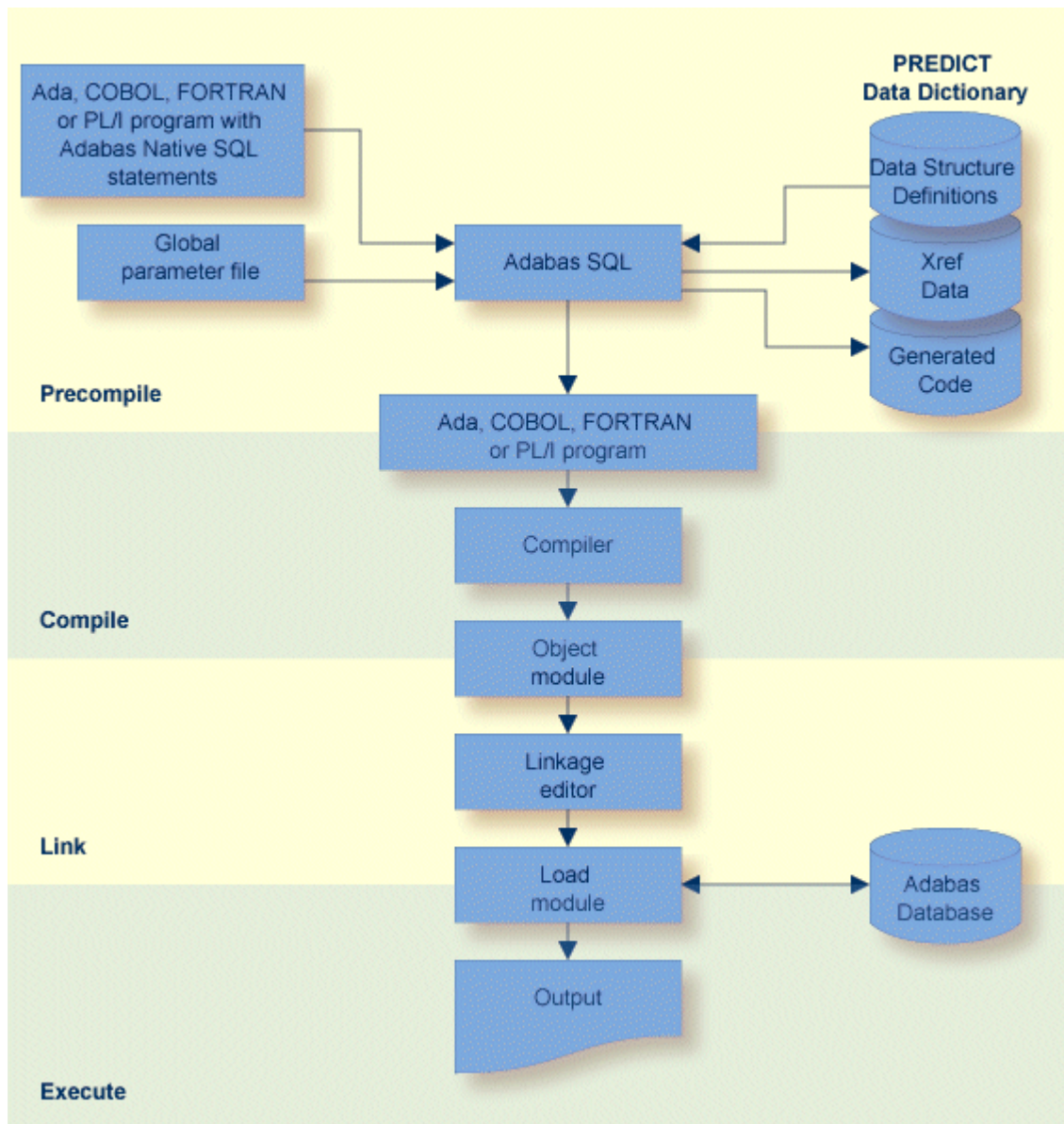
Consistent use of Adabas Native SQL throughout a data processing installation eliminates the risk of writing incorrect data declarations in programs that access the database. It also creates comprehensive records in the data dictionary that show which programs read from the database and which programs update it. This makes programs easier to maintain and provides the DBA with an effective management tool.

After it has been preprocessed by Adabas Native SQL, the program - containing data definitions and executable code generated by Adabas Native SQL as well as the original Ada, COBOL, FORTRAN or PL/I code written by the programmer - is compiled and link-edited in the normal manner.

2 PROGRAMMING CONSIDERATIONS

▪ Rules for Adabas Native SQL Statements	7
▪ Source Program Maintenance	8
▪ The Record Buffer and Reference to Data	9
▪ Response Code Interpretation	26
▪ Host Variables	28
▪ ISN Lists and the ISN Buffer	28
▪ HOLD Logic	29
▪ Security Options	30
▪ Record Buffer - ADA	31
▪ Record Buffer - COBOL	31
▪ Fields in FORTRAN	32
▪ Record Buffer - PL/I	33
▪ Date and Time Conversion Routines	34
▪ Support of Distributed Data Structures	35
▪ The Distribution handling	36
▪ Relational Null Support	37
▪ Long Alpha field Support	38

Using Adabas Native SQL does not require you to learn new programming techniques. Programs are written in Ada, COBOL, FORTRAN77 or PL/I as before, with Adabas Native SQL statements that access the Adabas database inserted at the required places. The Adabas Native SQL preprocessor converts the Adabas Native SQL statements into comments, inserts the generated code and data structures into the source stream and passes the remainder of the program through without alteration. At the same time, Adabas Native SQL optionally writes to the data dictionary a cross-reference list of the files and fields used by the program.



This chapter covers the following topics:

Rules for Adabas Native SQL Statements

Each Adabas Native SQL statement is preceded by "EXEC ADABAS". Each Adabas Native SQL statement is terminated by "END-EXEC" (in Ada, COBOL or FORTRAN), or by "END-EXEC" or ";" (in PL/I). These delimiters enable the preprocessor to distinguish Adabas Native SQL statements from regular Ada, COBOL, FORTRAN or PL/I code. The following COBOL program includes two Adabas Native SQL statements:

```
IDENTIFICATION DIVISION.
PROGRAM ID. EXAMPLE.
AUTHOR. SAG.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    SKIP2
    EXEC ADABAS
        BEGIN DECLARE SECTION
    END-EXEC
PROCEDURE DIVISION.
    EXEC ADABAS
        SELECT NAME, AGE, SALARY
        FROM PERSONNEL
        WHERE NUMBER-OF-DEPENDENTS GT 4
    END-EXEC
    DISPLAY NAME AGE SALARY
GOBACK.
```

"EXEC ADABAS" must be specified within one line. The same is true for "END-EXEC". Only one Adabas Native SQL statement may be written between "EXEC ADABAS" and "END-EXEC". The Adabas Native SQL statement is restricted to a maximum of 100 lines in length (including "EXEC ADABAS" and "END-EXEC").

Mixing Adabas Native SQL statements and regular source code statements is not allowed; Ada, COBOL, FORTRAN or PL/I code or comments should not appear between "EXEC ADABAS" and the corresponding "END-EXEC".



Note: (for COBOL users): The generated statements may include periods to terminate internal IF statements. Adabas Native SQL statements are therefore not permitted within IF...ELSE sections. This restriction does not apply to programs generated with the global parameter LANG COBOL/II or LANG COBOL/LE; in this case, Adabas Native SQL generates END-IF statements instead of periods, so there are no restrictions on nesting Adabas Native SQL statements within other IF...ELSE...END-IF statements.



Note: (for COBOL/II or COBOL/LE users): Adabas Native SQL will generate an extra statement with a period at the end while generating a SQL statement in COBOL/II in the case that the END-EXEC clause ends with a period: END-EXEC. In this case, users can “ask” Adabas Native SQL to generate a period at the end of the generation.

Source Program Maintenance

The source program stored in the programmer's library includes Adabas Native SQL statements, but not the code they generate. Therefore every compilation must be preceded by a pass through the Adabas Native SQL preprocessor. The preprocessor produces as its output a program in Ada, COBOL, FORTRAN or PL/I, including the original Adabas Native SQL statements, which are now marked as comments. This program should now be compiled and link-edited in the normal manner. In the compiler listing, the generated statements are identified in columns 73..80 by the characters "ADABAS" (executable code and internal data) or "ADADATA" (data definitions that are of use to you). This identification enables you to locate the lines that contain the data definitions easily.



Note: Do not alter variables that are declared in lines marked "ADABAS". You should only use those variables that are declared in lines marked "ADADATA".

Ada, FORTRAN and IBM PL/I source files may include line numbers in columns 73..80. COBOL source files may include line numbers in columns 1..6 and/or 73..80. Adabas Native SQL preserves this line-numbering, which serves as a cross-reference between the source code in the programmer's library and the compiler listing. The line sequence numbers are also used by the response code interpretation report and the TRACE report to help you when debugging.

PL/I source files in VMS environments may not include line numbers.

If the source code is not numbered, Adabas Native SQL automatically generates line numbers in columns 73..80.

The first Adabas Native SQL statement in the program must be the following:

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

Adabas Native SQL generates all the variables including the Adabas buffers after this statement.

Note for COBOL users: This statement must be in the WORKING-STORAGE SECTION of the DATA DIVISION.

The Record Buffer and Reference to Data

A record buffer is an area of storage in the user's program that is used by Adabas to transfer information to or from the database. Whenever an Adabas read command is executed, the desired database fields are located and copied into the record buffer.



Note: No record buffer is generated for FORTRAN programs; however, there is a character string which encompasses all fields and serves the same purpose as a record buffer.

Throughout this document, the term *record buffer* is used; however if a FORTRAN program is being discussed, this term should be interpreted as the character string referred to above.

Referencing Database Fields

To use data in database fields, refer to it using qualified identifiers composed of the record buffer name together with the basic field name as defined in the data dictionary. See table below.

Language	Form of Reference
Ada, PL/I	BUFFER.FIELD
COBOL	FIELD OF BUFFER
FORTRAN	No qualification possible



Note: If more than one database field is used, a prefix or suffix (in the SELECT statement itself) should be used to make the name unique.

If the Adabas Native SQL statement that causes the record buffer to be generated does not have an alias name in the FROM clause, then the level-1 record buffer name is the same as the (first) file name. If the FROM clause does include an alias name, then the alias name is used as the level-1 record buffer name. Levels are not used in Ada or FORTRAN.

Adabas Native SQL generates a name at level 2 for internal use only. Do not use this name in your programs.

Synonyms

The field names are generated beginning at level 3. The variable names that Adabas Native SQL generates are taken from Predict. If the program is written in Ada and an Ada field name synonym is defined in the data dictionary, then the synonym is used to generate the field name in the Adabas Native SQL record buffer. If the program is written in COBOL, FORTRAN or PL/I, then the COBOL, FORTRAN or PL/I field name synonym is used respectively. If no field name synonym is defined for the language in which the program is written, the basic name of the field is used. Note that the cross-reference information written to the data dictionary by Adabas Native SQL is always the basic name of the field and not the language-dependent synonym.

Prefix/Suffix

Having selected the field name or synonym, Adabas Native SQL then attaches the prefix and suffix to the name. These are taken from one of the following sources:

Source	Description
Local (highest priority)	Use the PREFIX and SUFFIX options for the current COMPARE, FIND, HISTOGRAM, INSERT, READ, SORT or UPDATE statement.
Global	Use the PREFIX and SUFFIX clauses of the global Adabas Native SQL OPTIONS parameter (see page)
Predict (lowest priority)	The current generation defaults for the respective language are used.

The first two options can only be used if the appropriate field in the Predict Modify...Defaults screen for Ada, COBOL, FORTRAN or PL/I is marked with an "X", indicating it may be modified by the user. Otherwise the prefix and suffix values defined in the data dictionary cannot be overridden.

Validation

The field name is now validated by examining it for characters that do not conform with the rules for forming identifiers in the appropriate language (Ada, COBOL, FORTRAN or PL/I). If any illegal characters are found, they are processed according to the setting of the 'validation character'. See table below:

Validation Character	Result
Null string (two consecutive apostrophes) in global parameter or Blank (Predict default)	Invalid characters in a field name will result in an error message but will not be modified.
Replace character (letters A-Z, digits 0-9 or special character depending on language)	Invalid characters in a field name are replaced by this character.
Asterisk	Invalid characters in the field name are deleted.

The validation character is taken from one of the following sources:

Source	Description
Global (higher priority)	Use the VALIDATION clause of the global OPTIONS parameter of Adabas Native SQL. Only possible if the field Val idate in the Predict Modify...Defaults screen is marked with an "X".
Predict	The current generation default for the respective language is used.

Truncation

If the field name is now too long, it is truncated by deleting characters from the left, middle or right, and a warning message is issued. The truncation character is taken from one of the following sources:

Source	Description
Global	Use the TRUNCATION clause of the global Adabas Native SQL <code>OPTIONS</code> parameter. Only possible if the field Truncation in the Predict Modify...Defaults screen is marked with an "X".
Predict	The current generation default for the respective language is used.

Field Attributes

The attributes of the variables (format, length, etc.) are also taken from the data dictionary. If the definition does not conform to the Ada, COBOL, FORTRAN or PL/I standards, the field is declared as an alphanumeric field. (Examples of non-conforming definitions would be 3 bytes binary or 5 bytes binary.)

Example: If there are fields called NAME and CITY in the Adabas file PERSONNEL, the following Adabas Native SQL statement-fragment is valid:

```
SELECT NAME, CITY
FROM PERSONNEL
```

You may refer to the variables in the record buffer as:


```
PERSONNEL.NAME, PERSONNEL.CITY           (Ada)
NAME OF PERSONNEL, CITY OF PERSONNEL     (COBOL)
NAME, CITY                               (FORTRAN)
PERSONNEL.NAME, PERSONNEL.CITY           (PL/I)
```


If you use the alias name option:

```
SELECT NAME, CITY
FROM PERSONNEL PERSON-ALIAS
```

then Adabas Native SQL generates a record buffer structure with the name PERSON_ ALIAS (Ada, PL/I) or PERSON-ALIAS (COBOL). You may refer to the variables in the record buffer as:

```
PERSON_ALIAS.NAME, PERSON_ALIAS.CITY      (Ada)
NAME OF PERSON-ALIAS, CITY OF PERSON-ALIAS (COBOL)
NAME, CITY                                (FORTRAN)
PERSON_ALIAS.NAME, PERSON_ALIAS.CITY      (PL/I)
```

 **Note:** (for FORTRAN users): Qualification is not possible in FORTRAN. However, if the database field is used in more than one Adabas Native SQL statement, a prefix or suffix (in the statement itself) must be used to make the name unique.

 **Note:** (for Ada and FORTRAN users): Numeric fields are transformed into character fields; therefore, whenever these fields are initialized and whenever values are assigned to these fields, the values must be filled with leading zeros, for example, "0001".

Groups

If the name specified is the name of a group (GR), Adabas Native SQL automatically generates declarations for the lower-level fields at all levels, in accordance with the definition stored in the data dictionary. The field names will be the full field names as defined in the data dictionary. If Ada, COBOL, FORTRAN or PL/I synonyms are defined in the data dictionary, they will be used in place of the full field names.

Example:

```
SELECT PERSON
FROM PERSONNEL
```

The structure of the Ada record buffer is as follows:

```
type RECORD_BUFPERs is
  record
    NAME           : STRING (1..20);
    FIRST_NAME     : STRING (1..15);
    INITIAL        : STRING (1..1);
    SEX            : STRING (1..1);
    AGE            : STRING (1..2);
    FAMILY_STATUS  : STRING (1..10);
    NUMBER_OF_DEPENDENTS : STRING (1..2);
    ISN            : INTEGER;
    QUANTITY       : INTEGER;
    RESPONSE_CODE  : SHORT_INTEGER;
  end record;
PERSONNEL: RECORD_BUFPERs;
```

The structure of the COBOL record buffer is as follows:

```

01 PERSONNEL.
  02 RECORD-BUF-0-1.
    03 PERSON.
      04 NAME PIC X(20).
      04 FIRST-NAME PIC X(15).
      04 INITIAL PIC X(1).
      04 P-DES.
        05 SEX PIC X(1).
        05 AGE PIC 9(2).
        05 FAMILY-STATUS PIC X(10).
        05 NUMBER-OF-DEPENDENTS PIC 9(2).
    02 ISN PIC 9(9) COMP.
    02 QUANTITY PIC 9(9) COMP.
    02 RESPONSE-CODE PIC 9(4) COMP.

```

The FORTRAN equivalent is as follows:

```

CHARACTER* 20 NAME
CHARACTER* 15 FNAME
CHARACTER* 1 INIIAL
CHARACTER* 1 SEX
CHARACTER* 2 AGE
CHARACTER* 10 FAMSTA
CHARACTER* 2 NUMNTS
CHARACTER* 51 PERSON
CHARACTER* 15 PDES
CHARACTER* 51 PERNEL

```



Note: Synonyms are assumed to be defined in the data dictionary as shown in Appendix B and truncation is assumed to occur in the middle of the word. (The maximum length of names depends on the operating system.)



Note: The field PERNEL encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

The structure of the PL/I record buffer is as follows:

```

DCL 1 PERSONNEL,
  2 RECORD_BUFFER_1 UNAL,
  3 PERSON,
    4 NAME CHAR (20),
    4 FIRST_NAME CHAR (15),
    4 INITIAL CHAR (1),
    4 P_DES,
      5 SEX CHAR (1),
      5 AGE PIC '(1)99',
      5 FAMILY_STATUS CHAR (10),

```

```

      5 NUMBER_OF_DEPENDENTS    PIC '(1)99',
      2 ISN                     FIXED BIN(31),
      2 QUANTITY                FIXED BIN(31),
      2 RESPONSE_CODE           FIXED BIN(15),
      RECORD_BUFPERS CHAR(51) BASED (ADDR(RECORD_BUFPERS_1));

```

Any field within a group may also be specified as a single field name.



Note: The level-2 name generated for the record buffer includes the cursor-name, if one was specified. The COBOL example shows a record buffer that was generated from an Adabas Native SQL statement without a cursor-name; the Ada and PL/I examples show a record buffer that was generated from an Adabas Native SQL statement with the cursor-name PERS.

Multiple-Value Fields

A multiple-value (MU) field is specified as a single field name; Adabas Native SQL takes the number of occurrences from the data dictionary. If the number of occurrences is specified as zero in the data dictionary, then Adabas Native SQL will declare 191 occurrences of the field. It is therefore strongly recommended that the number of occurrences be correctly specified in the data dictionary.

A single occurrence or a range of occurrences may optionally be specified within parentheses. The upper limit of the range or the number of the occurrence must not be greater than the number of occurrences as specified in the data dictionary, otherwise it will be ignored and a warning message will be printed. The valid formats are:

```
mu
```

```
mu(i)
```

```
mu(:var)
```

```
mu(i-j)
```

```
mu(LAST)
```

```
mu(i-LAST) (only at the end of the SELECT list)
```

where *mu* denotes the name of the multiple field; *i* and *j* denote integer constants; and *var* denotes the name of an integer variable. In Ada, *var* must be defined as "STRING(1..5)". In FORTRAN, *var* must be defined as "CHARACTER*5" and should contain a 5-digit number. LAST may be specified as the occurrence of an MU field to indicate that the last occurrence is to be read. For MU fields it is also possible to specify (*i*-LAST) at the end of the SELECT list to indicate a range of occurrences, from the occurrence with number *i* through to the last occurrence.

If a multiple-value field is referenced in the WHERE clause of a data retrieval statement, the only valid format is:

```
mu
```

If a single occurrence or a range not starting from 1 is specified, the name in the record buffer will be followed by a "-" or "_" and the number of the occurrence or the range.

Example:

```
SELECT OIL-CREDIT(1-5), OIL-CREDIT(7), OIL-CREDIT(9-10)
FROM FINANCE
```

The structure of the Ada record buffer is as follows:

```
type OIL_CREDITPERS is array (INTEGER range <>)
    of STRING (1..7);
type OIL_CREDIT_9_10PERS is array (INTEGER range <>)
    of STRING (1..7);


type RECORD_BUFFERS is
    record
        OIL_CREDIT      : OIL_CREDITPERS (1..5);
        OIL_CREDIT_7    : STRING (1..7);
        OIL_CREDIT_9_10 : OIL_CREDIT_9_10PERS (1..2);
        ISN              : INTEGER;
        QUANTITY         : INTEGER;
        RESPONSE_CODE   : SHORT_INTEGER;
    end record;
FINANCE : RECORD_BUFFERS;
```


The structure of the COBOL record buffer is as follows:

```
01 FINANCE.
02 RECORD-BUFFERS.
03 OIL-CREDIT      PIC X(7) OCCURS 5.
03 OIL-CREDIT-7    PIC X(7).
03 OIL-CREDIT-9-10 PIC X(7) OCCURS 2.
02 ISN              PIC 9(9) COMP.
02 QUANTITY         PIC 9(9) COMP.
02 RESPONSE-CODE   PIC 9(4) COMP.
```

The FORTRAN equivalent is as follows:

```
CHARACTER*      7  OCRE   (00005)
CHARACTER*      7  OCRE7
CHARACTER*      7  OCR910 (00002)
CHARACTER*     56  FINNCE
```

 **Note:** Synonyms are assumed to be defined in the data dictionary as shown in [Appendix B](#) and truncation is assumed to occur in the middle of the word. (The maximum length of names depends on the operating system.)

 **Note:** The field FINNCE encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

The structure of the PL/I record buffer is as follows:

```
DCL 1 FINANCE,
  2 RECORD_BUFPER_1 UNAL,
  3 OIL_CREDIT      (5) CHAR (7),
  3 OIL_CREDIT_7    CHAR (7),
  3 OIL_CREDIT_9_10 (2) CHAR (7),
  2 ISN             FIXED BIN(31),
  2 QUANTITY        FIXED BIN(31),
  2 RESPONSE_CODE   FIXED BIN(15),
  RECORD_BUFPER CHAR(56) BASED (ADDR(RECORD_BUFPER_1));
```

If the range is not explicitly specified, the default range is from the first occurrence up to the number specified in the data dictionary file (or 191 if the number of occurrences is not specified in the data dictionary).

In conjunction with multiple-value fields, you may additionally code *mu*(COUNT), i.e., the field name followed by the keyword COUNT in parentheses. This causes Adabas Native SQL to generate a special field in which Adabas stores the actual number of occurrences in the record. The field is two bytes long and has the following binary format:

- SHORT_INTEGER in ADA;
- PIC S9(4) COMP in COBOL;
- INTEGER*2 in FORTRAN;
- FIXED BIN(15,0) in PL/I.

The name generated for the COUNT field is the same as the name of the multiple-value field, preceded by:

- "C_" in ADA;
- "C-" in COBOL;
- "C" in FORTRAN;

■ "C_" in PL/I.

A count field is also generated if a count field is defined in a Predict field maintenance function. This is particularly useful in conjunction with the Adabas Native SQL `SELECT *` statement. A count field is never generated for a multiple-value field within a periodic group.

Example:

```
SELECT OIL-CREDIT, OIL-CREDIT(COUNT)
FROM FINANCE
```

The structure of the Ada record buffer is as follows:

```
type OIL_CREDITPERS is array (INTEGER range <>)
                        of STRING (1..7);
type RECORD_BUFERS is
  record
    OIL_CREDIT      : OIL_CREDITPERS (1..191);
    C_OIL_CREDIT    : SHORT_INTEGER;
    ISN              : INTEGER;
    QUANTITY         : INTEGER;
    RESPONSE_CODE   : SHORT_INTEGER;
  end record;
FINANCE: RECORD_BUFERS;
```

The structure of the COBOL record buffer is as follows:

```
01 FINANCE.
  02 RECORD-BUFERS.
    03 OIL-CREDIT PIC X(7) OCCURS 191.
    03 C-OIL-CREDIT PIC S9(4) COMP.
  02 ISN PIC 9(9) COMP.
  02 QUANTITY PIC 9(9) COMP.
  02 RESPONSE-CODE PIC 9(4) COMP.
```

The FORTRAN equivalent is as follows:

```
CHARACTER*      7 OCRE   (00191)
INTEGER*        2 COCRE
CHARACTER*     1340 FINNCE
```



Note: Synonyms are assumed to be defined in the data dictionary as shown in [Appendix B](#) and truncation is assumed to occur in the middle of the word. (The maximum length of names depends on the operating system.)



Note: The field FINNCE encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

The structure of the PL/I record buffer is as follows:

```
DCL 1 FINANCE,
  2 RECORD_BUFERS_1 UNAL,
  3 OIL_CREDIT (191) CHAR (7),
  3 C_OIL_CREDIT FIXED BIN(15,0),
  2 ISN FIXED BIN(31),
  2 QUANTITY FIXED BIN(31),
  2 RESPONSE_CODE FIXED BIN(15),
  RECORD_BUFERS CHAR(1339) BASED (ADDR(RECORD_BUFERS_1));
```

Periodic Groups

A periodic group (PE) consists of up to 65000 occurrences of a group. The default number of occurrences remains 99, as in the previous version. Adabas Native SQL automatically generates definitions of all fields within the periodic group, using the full field names as defined in the data dictionary, or the Ada, COBOL, FORTRAN or PL/I synonyms if present. You may limit the number of occurrences as for multiple value fields. A COUNT field containing the number of occurrences of the periodic group may be generated by coding *pe*(COUNT) or by defining a PE count field with a Predict field maintenance function. Valid formats:

pe

pe(*i*)

pe(:*var*)

pe(*i-j*)

where *pe* denotes the name of the periodic group; *i* and *j* denote integer constants; and *var* denotes the name of an integer variable. In Ada, *var* must be defined as "STRING(1..5)". In FORTRAN, *var* must be defined as "CHARACTER*5" and should contain a 5-digit number.

If a periodic group is referenced in the WHERE clause of a data retrieval statement, the valid formats are:

pe

pe(*i*)

Suffixes defining a single occurrence or a range of occurrences not starting from 1 will be added to all fields within the periodic group. A range starting from the first occurrence is not given a suffix.

If you do not need all the fields within the periodic group, you may request individual fields, which are treated as multiple-value fields, except that you may not request the COUNT of such a field, but only the COUNT of the periodic group as a whole.

For COBOL and PL/I, Adabas Native SQL supports the GROUP STRUCT attribute which can be defined in the data dictionary for periodic groups. Correct use of this attribute can result in a significantly shorter Adabas format buffer. For more information see *Defining More Attributes of Fields, 3GL Specification* in section *Field* of Chapter *Predefined Object Types* of the Predict Reference Manual.



Note: (for Ada and FORTRAN users): Periodic groups will always be generated with GROUP STRUCT = N, and no consideration will be given to the Predict definition.

Example:

```
SELECT MAJOR-CREDIT(1), MAJOR-CREDIT(3-5), MAJOR-CREDIT(7),
       MAJOR-CREDIT(COUNT)
FROM FINANCE
```

The structure of the Ada record buffer is as follows:

```
type CREDIT_CARD_3_5PERS is array (INTEGER range <>)
                                of STRING (1..18);
type CREDIT_LIMIT_3_5PERS is array (INTEGER range <>)
                                of STRING (1..4);
type CURRENT_BALANCE_3_5PERS is array (INTEGER range <>)
                                of STRING (1..4);
type RECORD_BUFPERs is
  record
    CREDIT_CARD_1      : STRING (1..18);
    CREDIT_LIMIT_1    : STRING (1..4);
    CURRENT_BALANCE_1 : STRING (1..4);
    CREDIT_CARD_3_5   : CREDIT_CARD_3_5PERS (1..3);
    CREDIT_LIMIT_3_5  : CREDIT_LIMIT_3_5PERS (1..3);
    CURRENT_BALANCE_3_5 : CURRENT_BALANCE_3_5PERS (1..3);
    CREDIT_CARD_7     : STRING (1..18);
    CREDIT_LIMIT_7    : STRING (1..4);
    CURRENT_BALANCE_7 : STRING (1..4);
    C_MAJOR_CREDIT    : SHORT_INTEGER;
    ISN               : INTEGER;
    QUANTITY          : INTEGER;
    RESPONSE_CODE     : SHORT_INTEGER;
  end record;
FINANCE: RECORD_BUFPERs;
```

The structure of the COBOL record buffer is as follows:

```

01 FINANCE.
  02 RECORD-BUFFERS.
    03 MAJOR-CREDIT-1.
      04 CREDIT-CARD-1      PIC X(18).
      04 CREDIT-LIMIT-1    PIC 9(4).
      04 CURRENT-BALANCE-1 PIC 9(4).
    03 G-MAJOR-CREDIT-3-5.
      04 MAJOR-CREDIT-3-5          OCCURS 3.
      05 CREDIT-CARD-3-5    PIC X(18).
      05 CREDIT-LIMIT-3-5   PIC 9(4).
      05 CURRENT-BALANCE-3-5 PIC 9(4).
    03 MAJOR-CREDIT-7.
      04 CREDIT-CARD-7      PIC X(18).
      04 CREDIT-LIMIT-7    PIC 9(4).
      04 CURRENT-BALANCE-7 PIC 9(4).
    03 C-MAJOR-CREDIT      PIC S9(4) COMP.
  02 ISN                   PIC 9(9) COMP.
  02 QUANTITY              PIC 9(9) COMP.
  02 RESPONSE-CODE        PIC 9(4) COMP.

```

The FORTRAN equivalent is as follows:

```

CHARACTER* 18 CCARD1
CHARACTER*  4 CLIM1
CHARACTER*  4 CBAL1
CHARACTER* 26 MAJIT1
CHARACTER* 18 CCAD35(00003)
CHARACTER*  4 CLIM35(00003)
CHARACTER*  4 CBAL35(00003)
CHARACTER* 78 MAJT35
CHARACTER* 18 CCARD7
CHARACTER*  4 CLIM7
CHARACTER*  4 CBAL7
CHARACTER* 26 MAJIT7
INTEGER*   2 CMADIT
CHARACTER* 132 FINNCE

```



Note: Synonyms are assumed to be defined in the data dictionary as shown in [Appendix B](#) and truncation is assumed to occur in the middle of the word. (The maximum length of names depends on the operating system.)



Note: The field FINNCE encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

The structure of the PL/I record buffer is as follows:

```

DCL 1 FINANCE,
  2 RECORD_BUFPEPERS_1 UNAL,
  3 MAJOR_CREDIT_1,
  4 CREDIT_CARD_1          CHAR(18),
  4 CREDIT_LIMIT_1        PIC '(3)99',
  4 CURRENT_BALANCE_1     PIC '(3)99',
  3 G_MAJOR_CREDIT_3_5,
  4 MAJOR_CREDIT_3_5      (3),
  5 CREDIT_CARD_3_5      CHAR(18),
  5 CREDIT_LIMIT_3_5     PIC '(3)99',
  5 CURRENT_BALANCE_3_5  PIC '(3)99',
  3 MAJOR_CREDIT_7,
  4 CREDIT_CARD_7        CHAR(18),
  4 CREDIT_LIMIT_7       PIC '(3)99',
  4 CURRENT_BALANCE_7    PIC '(3)99',
  3 C_MAJOR_CREDIT        FIXED BIN(15,0),
  2 ISN                   FIXED BIN(31),
  2 QUANTITY              FIXED BIN(31),
  2 RESPONSE_CODE         FIXED BIN(15),
  RECORD_BUFPEPERS CHAR(132) BASED(ADDR(RECORD_BUFPEPERS_1));

```

Multiple-Value Fields within Periodic Groups

Adabas Native SQL supports multiple-value fields that occur within periodic groups. If the number of occurrences is not specified, the number of occurrences is taken from the data dictionary. If the number of occurrences is not explicitly specified, or if the index is variable, the occurrence number is not appended as a suffix to the field name.

Reference to elements of such a field is made as follows:

mp

mp(i(k))

mp(i(k-1))

mp(i-j(k))

mp(i-j(k-1))

mp(:ivar(k))

mp(:ivar(k-1))

mp(i(:kvar))

mp(i-j(:kvar))

mp(:ivar(:kvar))

mp(LAST)

mp(LAST(LAST))

`mp(i(k-LAST))` (only at the end of the SELECT list)

`mp` denotes the name of the multiple-value field. `i`, `i-j` and `ivar` indicate which group or groups are required. `k`, `k-l` and `kvar` indicate which occurrence or occurrences of the multiple-value field are required. `i`, `j`, `k` and `l` denote integer constants. `j` must be greater than `i`, and both must be in the range 1..191. `l` must be greater than `k`, and both must be in the range 1..191. `ivar` and `kvar` denote the names of integer variables. LAST means the last occurrence.

If a multiple-value field within a periodic group is referenced in the WHERE clause of a data retrieval statement, the only valid format is:

`mp`

Counter fields can also be generated for multiple-value fields occurring within periodic groups. `mp(COUNT1)` generates a counter field containing the number of occurrences of the multiple-value field `mp` in the first occurrence of the periodic group, `mp(COUNT1-3)` generates counter fields for the multiple-value field `mp` in each of the first three occurrences of the periodic group, and `mp(COUNTLAST)` generates a counter field for the multiple-value field in the last occurrence of the periodic group. The names of the counter fields are:

ADA	COBOL	FORTRAN	PL/I
<code>C_mp_1</code>	<code>C-mp-1</code>	<code>Cmp1</code>	<code>C_mp_1</code>
<code>C_mp_2</code>	<code>C-mp-2</code>	<code>Cmp2</code>	<code>C_mp_2</code>
<code>C_mp_3</code>	<code>C-mp-3</code>	<code>Cmp2</code>	<code>C_mp_3</code>

Example:

```
SELECT INSURANCE-COMPANY(2-4(6-8))
FROM FINANCE
```

The structure of the Ada record buffer is as follows:

```
type INSURANCE_COMPANY_6_8PERS is array (INTEGER range <>,
                                         INTEGER range <>)
                                         of STRING (1..25);
type RECORD_BUFPERs is
  record
    INSURANCE_COMPANY_6_8 : INSURANCE_COMPANY_6_8PERS (1..3, 1..3);
    ISN                    : INTEGER;
    QUANTITY               : INTEGER;
    RESPONSE_CODE         : SHORT_INTEGER;
  end record;
FINANCE: RECORD_BUFPERs;
```

The structure of the COBOL record buffer is as follows:

```

01 FINANCE.
  02 RECORD-BUFFERS.
    03 A-INSURANCE-COMPANY-2-4          OCCURS 3.
      04 INSURANCE-COMPANY-6-8 PIC X(25) OCCURS 3.
  02 ISN                               PIC 9(9) COMP.
  02 QUANTITY                           PIC 9(9) COMP.
  02 RESPONSE-CODE                       PIC 9(4) COMP.

```

The FORTRAN equivalent is as follows:

```

CHARACTER* 25 INCM68(00003 , 00003)
CHARACTER* 225 FINNCE

```



Note: Synonyms are assumed to be defined in the data dictionary as shown in [Appendix B](#) and truncation is assumed to occur in the middle of the word. (The maximum length of names depends on the operating system.)



Note: The field FINNCE encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

The structure of the PL/I record buffer is as follows:

```

DCL 1 FINANCE,
  2 RECORD_BUFFERS_1 UNAL,
  3 A_INSURANCE_COMPANY_2_4 (3),
  4 INSURANCE_COMPANY_6_8 (3) CHAR(25),
  2 ISN                      FIXED BIN(31),
  2 QUANTITY                 FIXED BIN(31),
  2 RESPONSE_CODE           FIXED BIN(15),
  RECORD_BUFFERS CHAR(225) BASED(ADDR(RECORD_BUFFERS_1));

```

Additional Fields in the Record Buffers (Ada, COBOL, PL/I)

If a field is specified in the SELECT clause, and Predict contains redefinitions for this field, then the redefined fields are also included in the record buffer. The prefix and suffix are added to the field names and the result is truncated if necessary. (Ada does not support redefinition.)

Unless the global parameter `ABORT` is specified, Adabas Native SQL appends three fields to each record buffer. A record buffer containing these three fields is also generated for DELETE statements, although no database fields are generated. The names of the fields are shown in the tables below. They may only be used in conjunction with an adequate file name.

If the global parameter `ABORT` is specified, these three fields are generated as global data and they have the names `SQLISN`, `SQLQTY` and `SQLRSP`, as used in FORTRAN programs. Since no record buffers are ever generated for FORTRAN, the field names are always global to the program.

The ISN variable is a 4-byte binary field in which Adabas returns the ISN (internal sequence number) of the (first) record found or read or, in the case of a HISTOGRAM command where the descriptor is in a periodic group, the number of the current occurrence. The ISN variable is defined as:

Language	Variable Name*	Format
ADA	ISN	INTEGER
COBOL	ISN	PIC 9(9) COMP
PL/I	ISN	FIXED BIN (31)

* The variable name is SQLISN if the global parameter `ABORT .` is coded. See description of the **ABORT** parameter for more information.

The QUANTITY variable is a 4-byte binary field which, when used in conjunction with a COMPARE, FIND, FIND COUPLED or SORT statement, is available after executing the OPEN statement. It returns the number of ISNs in the ISN list, or the number of ISNs in the ISN buffer. When used in conjunction with a HISTOGRAM statement, the quantity variable, which is available after executing the FETCH statement, returns the number of records that contain the specified descriptor value. (The quantity variable is not available in conjunction with READ statements.) The quantity variable is defined as:

Language	Variable Name*	Format
ADA	QUANTITY	INTEGER
COBOL	QUANTITY	PIC 9(9) COMP
PL/I	QUANTITY	FIXED BIN (31)

* The variable name is SQLQTY if the global parameter `ABORT .` is coded. See description of the **ABORT** parameter for more information.

The RESPONSE_CODE (Ada), RESPONSE-CODE (COBOL) or RESPONSE_CODE (PL/I) variable is a 2-byte binary field in which Adabas returns the response code after execution of the command. The response code variable is defined as:

Language	Variable Name*	Format
ADA	RESPONSE_CODE	SHORT_INTEGER
COBOL	RESPONSE-CODE	PIC 9(4) COMP
PL/I	RESPONSE_CODE	FIXED BIN (15)

* The variable name is SQLRSP if the global parameter `ABORT .` is coded.

See *Response Code Interpretation* and the description of the **ABORT** parameter for more information.

Additional Fields in FORTRAN Programs

Adabas Native SQL enters values in three global variables after each SQL statement. These variables contain only the values generated by the last command and will be changed when a new command is issued.

The ISN variable is a 4-byte binary field in which Adabas returns the ISN (internal sequence number) of the (first) record found or read or, in the case of a HISTOGRAM command where the descriptor is in a periodic group, the number of the current occurrence.

Language	Variable Name	Format
FORTRAN	SQLISN	INTEGER*4

The QUANTITY variable is a 4-byte binary field which, when used in conjunction with a COMPARE, FIND, FIND COUPLED or SORT statement, is available after executing the OPEN statement. It returns the number of ISNs in the ISN list, or the number of ISNs in the ISN buffer. When used in conjunction with a HISTOGRAM statement, the quantity variable, which is available after executing the FETCH statement, returns the number of records that contain the specified descriptor value. (The quantity variable is not available in conjunction with READ statements.)

Language	Variable Name	Format
FORTRAN	SQLQTY	INTEGER*4

The response code variable is a 2-byte binary field in which Adabas returns the response code after execution of the command.

Language	Variable Name	Format
FORTRAN	SQLRSP	INTEGER*2

See [Response Code Interpretation](#) and the description of the **ABORT** parameter for more information.

If you want to use for example the response codes returned by more than one statement, then you must save each response code before new SQL statements are executed.

End-of-File Flag (ADACODE, SQLCOD)

The ADACODE (Ada, COBOL, DEC FORTRAN and PL/I) or SQLCOD (IBM FORTRAN) variable is a 2-byte binary field in which Adabas Native SQL returns an end-of-file flag. The value 3 in this field indicates that end-of-file was detected in a sequential read command, or end-of-list after reading all the records found by a search statement. It is defined as:

Language	Variable Name	Format
ADA	ADACODE	SHORT_INTEGER
COBOL	ADACODE	PIC 9(4) COMP
FORTRAN	SQLCOD	INTEGER*2
FORTRAN/VMS	ADACODE	INTEGER*2
PL/I	ADACODE	FIXED BIN (15)

Response Code Interpretation

The Adabas response code is a code that is returned to the caller after every Adabas command. It is stored in a variable called RESPONSE-CODE (COBOL) or RESPONSE_CODE (Ada and PL/I) in the record buffer of the command that was executed, or in the global variable SQLRSP (FORTRAN). A value of zero returned in this variable indicates that the Adabas Native SQL statement has been executed successfully. A non-zero value (other than 3, which denotes end-of-file) indicates that an error occurred. In this case, the statement has not been executed. Each value is associated with a distinct type of error, as shown in the list below.

Adabas Native SQL automatically calls an error-checking routine after each Adabas command if the response code is non-zero. Software AG supplies default routines which check and interpret the response code. If the response code has a value other than 3, the routine prints out the appropriate error message, the contents of the Adabas control block and the line number of the erroneous statement in the source program, calls an appropriate trace module, issues a `backout transaction` (ROLLBACK WORK) command, closes the database (DBCLOSE), and finally terminates the program.

Language	Default Abort Module	Default Trace Module
ADA	RESPF	PRTRAC
COBOL	RESPINT	PRTRACE
FORTRAN	RESPF	PRTRAC
PL/I	RESPINT	PRTRACE

In many cases, the action described above may be all that is required. However, if the action taken by the standard routine is inappropriate or insufficient, the ABORT parameter can be used to specify that a user-defined error handling routine with a different name should be called instead.

The data administrator will know whether alternative error handling routines are available at your installation.

See also the description of the **ABORT** parameter.

Response Codes

The response code is returned in the variable `RESPONSE_CODE` (Ada), `RESPONSE-CODE` (COBOL), `SQLRSP` (FORTRAN) or `RESPONSE_CODE` (PL/I) that is attached to every record buffer. The normal response code (success) is 0.

If the following response code occurs and the error handling routine is that shown in the table above, control will be returned to the user program directly following the statement that caused the response code.

Response code	Meaning
3	Response Code 3 (which is also signaled in the variable <code>ADACODE</code> (Ada, COBOL or PL/I) or <code>SQLCOD</code> (FORTRAN)) indicates that end-of-file was detected in a sequential read command, or end-of-list after reading all the records found by a search statement.

The following response codes may also occur during normal operation. If a user-written error handling routine is called, it should take appropriate action for all response codes that might occur. This might include printing an error message and/or returning to the application program. The standard error handling routines `RESPINT` and `RESPF` supplied by Software AG can be used as a model when writing this routine.

Response code	Meaning
1	The ISN list is too big to be sorted.
9	A partially-completed transaction has been automatically backed-out, possibly as the result of a timeout (for programs that use ET-mode). Note that Adabas may release the command-ID when Response Code 9 occurs. ISN lists, hold queue entries and user data (see also the <code>CHECKPOINT</code> , <code>COMMIT WORK</code> , <code>CONNECT</code> , <code>DBCLOSE</code> and <code>READ USERDATA</code> statements) are no longer accessible.
17	Invalid file number. A file required by the program could not be found in the database
19	An attempt has been made to update a file that was opened for access only
41	Adabas has detected an error in the format buffer. This can be caused by an incorrect data field definition in Predict.
48	The user-ID specified in the <code>CONNECT</code> statement is already in use; or the mode of usage specified for a file in the <code>CONNECT</code> statement conflicts with the file's current usage.
98	A descriptor value in a record to be INSERTed or UPDATEd exists already in the file and the file has the 'unique descriptor' attribute (VAX response code).
113	A <code>READ ISN</code> statement without the <code>SEQUENCE</code> option was issued and Adabas could not find a record having the specified ISN; or a <code>READ ISN</code> statement attempted to read a record

Response code	Meaning
	and the 'security by value' check failed. It can also indicate that an INSERT statement using the WHERE ISN=n clause specified an ISN that was already present in the file.
144	An UPDATE or DELETE statement was issued but the relevant record was not in hold status for the program that issued the statement.
145	The program attempted to hold a record that is already being held by another user. This code may be returned if the HOLD RETURN option is used.
148	The Adabas nucleus is not available.
198	A descriptor value in a record to be INSERTed or UPDATEd exists already in the file and the file has the 'unique descriptor' attribute.

See section *Adabas Response Codes* in the *Adabas Messages and Codes Manual* for more information.

Host Variables

Host variables are normal program variables that are also used in Adabas Native SQL statements. They are declared using normal Ada, COBOL, FORTRAN or PL/I statements. When used in an Adabas Native SQL statement, the name of each host variable must be immediately preceded by a colon (":"), for example ":NAME".

ISN Lists and the ISN Buffer

The abbreviation ISN occurs frequently in this manual. It stands for Internal Sequence Number: a reference number that identifies each record uniquely within an Adabas file. Each new record created by the INSERT statement must have an ISN. If you do not allocate the ISN explicitly, it is assigned automatically by Adabas. When allocating ISNs, care should be taken that each ISN is unique and that no ISN that exceeds the MAXISN parameter is specified.

When a FIND statement finds more than one record in the file, Adabas makes a list of the ISNs of these records and returns this ISN list as the result of the FIND operation.

You have the option of providing an ISN buffer, whose size is specified by the ISNSIZE parameter either in the global **OPTIONS** parameter or in each individual Adabas Native SQL statement. If an ISN buffer of adequate size is provided, Adabas stores the ISN list in this buffer. If an ISN buffer is not provided, or if it is too small to contain the ISN list created by a particular FIND statement, then the excess ISNs are automatically written to the Adabas workfile. They are then read from the ISN buffer and/or from the workfile and returned to the user one by one each time a statement (for example, FETCH) that requires an ISN is executed.

In general, programs run more efficiently if the ISN buffer is large enough to contain the entire ISN list. However, if the ISN buffer has to be made smaller, the program will continue to run exactly as before; the process of buffering excess ISNs in the Adabas workfile is completely transparent to the user.

The ISN buffer cannot be used if Adabas security by value is in effect, or in CICS or UTM programs that use the Adabas Native SQL statements `SAVE` and `RESTORE`.

HOLD Logic

The `HOLD` option can be used with all Adabas Native SQL data retrieval statements except `HISTOGRAM` to place the record in hold status. A record in hold status is prevented from being updated by other users until it is explicitly released by issuing a `COMMIT WORK`, `ROLLBACK WORK` or `RELEASE` statement. This avoids the conflict that would arise if two or more users attempted to update one record simultaneously.

RETURN Option

The presence or absence of the `RETURN` option determines Adabas's response if the record to be accessed is currently being held by another user.

If `HOLD` is used without the `RETURN` option and an attempt is made to access a record held by another user, the program is suspended until the record is released by the other user.

If `HOLD` is used with the `RETURN` option and an attempt is made to access a record held by another user, Adabas returns Response Code 145 to the user program. If the response code interpretation routine as supplied by Software AG is being used, an error message is printed and the program ABENDs. If some other action is required, an alternative routine that checks for this response code and takes appropriate action must be supplied (see also the description of the [ABORT](#) parameter). The response code is returned in the variable `RESPONSE_CODE` (Ada), `RESPONSE-CODE` (COBOL) or `RESPONSE_CODE` (PL/I), which is attached to every record buffer, or in the global variable `SQLRSP` (FORTRAN).

See section *Competitive Updating* in the Adabas Command Reference Manual for more information on Adabas hold logic.

Security Options

Adabas offers the following facilities to prevent unauthorized users from accessing or updating confidential data:

- Password protection
- Ciphering
- Security by value.

Password Protection

Password protection permits only those database operations that cite the correct password. Adabas commands that include an incorrect password, or no password at all, are rejected. Furthermore, access and update security levels are associated with each password. Whenever a database operation is executed, Adabas checks that the security level associated with the password equals or exceeds the security level of the database, both at the file level and at the field level. Password protection therefore provides a very flexible mechanism for controlling the degree of access individual computer users can exercise.

Ciphering

If a file is ciphered, the data are stored on disk in an encrypted format that is incomprehensible to any user who does not know the correct cipher key. Adabas uses the cipher key in conjunction with a special decryption algorithm to reconstruct the original data. Cipher protection offers a very high level of security against unauthorized efforts to read data from a database. Conversely, a file update made with a wrong cipher key is conspicuous because the decryption algorithm converts the data into a meaningless jumble when a legitimate user tries to read them.

Further details of the password and data encryption security facilities are given in the section *Security Planning* in the *Adabas DBA Reference Manual*.

Security by Value

The third security option Adabas offers is security by value. Using this facility, access to records is controlled by the values contained in specified fields. For example, a user may be forbidden from accessing records in the PERSONNEL file that have a value in the SALARY field exceeding 6000.

The ISNSIZE option cannot be used when processing files that are protected by this feature. See page for more information.

See the *Adabas Security Manual* for more information. Note that this manual is only sent to DBAs on written application.

Consult your DBA before writing programs that access files protected by any of the mechanisms described in this section.

Record Buffer - ADA

The fields generated in the record buffers in Ada programs have the clauses shown in the table below:

Predict Format	Predict Length	Ada clause	Observations
A	<i>nnn</i>	STRING (1.. <i>nnn</i>)	
B or I	1	SHORT_SHORT_INTEGER	VMS only
B or I	2	SHORT_INTEGER	
B or I	4	INTEGER	
F	4	FLOAT	
F	8	LONG_FLOAT	VMS only
N or U	<i>nn.m</i>	STRING (1.. <i>nn+m</i>)	
P	<i>nn.m</i>	STRING (1.. <i>y</i>)	
L		BOOLEAN	
D		STRING (1..4)	
T		STRING (1..7)	
Counter fields	SHORT_INTEGER		



Note: Numeric fields are transformed into character fields; therefore, whenever these fields are initialized and whenever values are assigned to these fields, the values must be filled with leading zeros, for example "0001".



Note: $y = (nn+m+1) / 2$

Record Buffer - COBOL

The fields generated in the record buffers in COBOL programs have the clauses shown in the table below:

Predict Format	Predict Length	COBOL clause	Observations
A	nnn	PIC X(nnn)	
B or I	2	PIC S9(4) COMP	
B or I	4	PIC S9(9) COMP	
B or I	8	PIC S9(18) COMP	
F	4	COMP-1	
F	8	COMP-2	
N or U	nn.m	PIC 9(nn)V9(m)	In any of these fields, nn+m may not exceed 18, and if m=0 the term V9(m) is omitted
NS or US	nn.m	PIC S9(nn)V9(m)	
P	nn.m	PIC 9(nn)V9(m)COMP-3	
PS	nn.m	PIC S9(nn)V9(m)COMP-3	
L		PIC X	
D		PIC 9(7) COMP-3	
T		PIC 9(13) COMP-3	
Counter fields		PIC S9(4) COMP	

An automatically generated counter field has the clause PIC S9(4) COMP.

A numeric or binary format field with a length not included in the table above is treated in COBOL as an alphanumeric format field

Packed fields in COBOL/II under operating system BS2000/OSD are generated as "PACKED DECIMAL" instead of "COMP-3".

No alignment is performed.

Fields in FORTRAN

The fields generated in FORTRAN programs have the clauses shown in the table below:

Predict		FORTRAN Clause	Compiler	Alignment assuming word length=4
Format	Length			
A	nnn	CHARACTER*nnn	any	
B or I	1	LOGICAL*1	IBM, Siemens, VMS	
B or I	2	INTEGER*2	IBM, Siemens, VMS	half-word boundary
B or I	4	INTEGER*4	IBM, Siemens, VMS	word boundary
B or I	8	INTEGER*8	Siemens	double-word boundary
B or I	8	CHARACTER*8	IBM, VMS	

Predict		FORTRAN Clause	Compiler	Alignment assuming word length=4
Format	Length			
F	4	REAL*4	IBM, Siemens, VMS	word boundary
F	8	REAL*8	IBM, Siemens, VMS	double-word boundary
N or NS, U or US	nn.m	CHARACTER*x where $x=nn+m$	any	
P or PS	nn.m	CHARACTER*y where $y=(nn+m+1)/2$	any	
L		LOGICAL*1	any	
D		CHARACTER*4	any	
T		CHARACTER*7	any	

- If generated for IBM, Siemens or VMS compilers: Any file number field, length fields and automatically generated counter fields have the clause INTEGER*2.



Note: Numeric fields are transformed into character fields; therefore, whenever these fields are initialized and whenever values are assigned to these fields, the values must be filled with leading zeros, for example "0001".

Record Buffer - PL/I

The fields generated in the record buffers in PL/I programs have the clauses shown in the table below:

Fields in the PL/I include code have a PL/I clause determined by the length and format of the corresponding Predict field object, as shown in the table below where s is the numeric sign whose content (T , I , or $9R$) and position (left or right) are defined in the PL/I generation defaults; $nn+m$ must not exceed 15; and if m is zero, $V(m)9$ is omitted.

Predict		PL/I clause	Observations
Format	Length		
A	nnn	CHAR (nnn)	
B	1	FIXED BIN(7)	VMS only
B or I	2	FIXED BIN (15,0)	
B or I	4	FIXED BIN (31,0)	
F	4	FLOAT DEC (6)	
F	8	FLOAT DEC (16)	
N or U	nn.m	PIC '(nn)9V(m)9'	

Predict		PL/I clause	Observations
Format	Length		
NS or US	nn.m	PIC 's(nn-1)9V(m)9' or PIC '(nn)9V(m-1)9s'	
P or PS	nn.m	FIXED DEC (nn+m,m)	
L		BIT (8)	
D		FIXED DEC (7,0)	
T		FIXED DEC(13,0)	
Counter fields		FIXED BIN (15,0)	

A numeric or binary format field with a length not included in the table above is treated in PL/I as an alphanumeric format field.

Date and Time Conversion Routines

The following routines are delivered with this version of Adabas Native SQL and can be used in the application:

- SQTODATE
- SQFRDATE
- SQTOTIME
- SQFRTIME

SQTODATE

This module accepts two parameters:

- N-DATE (N8) in format DDMMYYYY
- DATE (D)

It converts the first parameter into a format D number and returns it in the second parameter.

SQFRDATE

This module accepts two parameters:

- N-DATE (N8) in format DDMMYYYY
- DATE (D)

It converts the second parameter, which is a format D number, into a numeric date and returns it in the first parameter.

SQTOTIME

This module accepts three parameters:

- N-DATE (N8) in format DDMMYYYY
- N-TIME (N7) in format HHMMSS
- TIME (T)

It converts the first and second parameters into a format T number and returns it in the third parameter.

SQFRTIME

This module accepts three parameters:

- N-DATE (N8) in format DDMMYYYY
- N-TIME (N7) in format HHMMSS
- TIME (T)

It converts the third parameter, which is a format T number, into a numeric date and numeric time and returns them in the first and second parameters.

Support of Distributed Data Structures

Adabas Native SQL supports distributed data structures by the DBID or AUTODBID clauses in Adabas Native SQL statements, or the Global OPTIONS parameters AUTODBID-ALL, AUTODBID-ATM, AUTODBID and DBID. These clauses put the DBID number defined in Predict in the control block.

- [The Global Parameters NETWORK and VIRTUAL-MACHINE](#)

The Global Parameters NETWORK and VIRTUAL-MACHINE

These global parameters are mandatory if more than one network is defined in Predict.

These parameters define the network and virtual machine in which the program is to run. Adabas Native SQL checks that the network and virtual machine exist in Predict and that the virtual machine is linked as a child object to the network.

For every database used (DBID, AUTODBID, AUTODBID-ATM and AUTODBID-ALL clauses) Adabas Native SQL checks the following:

- that if the database is defined as local, it is linked to the current virtual machine,

- that if the database is defined as isolated, it is linked (via the current virtual machine) to the current network.



Note: In this section, the terms *current network* and *current virtual machine* are used to describe the network and virtual machine specified with the global parameters **NETWORK** and **VIRTUAL-MACHINE** respectively.

The Distribution handling

The distribution is handled by the application programmer. If the program uses the DBID, AUTODBID, AUTODBID-ATM and AUTODBID-ALL, Adabas Native SQL performs the following additional checks:

- If one of the DBID clauses is used, the Run Mode parameter of the corresponding Predict database object must be *I* (isolated) or *L* (local), otherwise an error message is given.
- If the database is *local*, Adabas Native SQL checks that it is linked to the current virtual machine.
- If the database is *isolated*, Adabas Native SQL checks that it is linked to the current network.

After checking the database, Adabas Native SQL checks the physical link between the file and the database. The physical link information is stored in the Adabas attributes in Predict for every physical file connected to the database. This information includes the physical file number and the physical Logical Distribution type (how the file is implemented). This type must be either *blank* (simple file) or *E* (expanded).

If the file is expanded, this means that there are several files with the same layout in the same database, and that every file has a different range of ISNs. Adabas Native SQL checks for the physical file with the lowest minimum ISN value (ADALOD LOAD parameter MINISN).

With both simple and expanded files, Adabas Native SQL takes the physical file number from this physical link information. Note that in previous versions of Predict, the physical file number and the logical file number (as exists in the file description) had to be identical. As of Predict Version 3.2 or above, however, the same logical file may have different physical file numbers.

With this kind of distribution, the application is responsible for defining the DBID where every file exists. The AUTODBID-ALL option allows an update program which updates one database and accesses up to five more databases. With AUTODBID-ALL, Adabas Native SQL automatically detects which is the updated database and issues the COMMIT and ROLLBACK commands to it. It also generates different CONNECT and DBCLOSE statements to the different databases.

There is another option AUTODBID-ATM that may be used only in cases that the application will run under the control of the Adabas Transaction Manager (ATM) . With this option Adabas Native SQL does not restrict the number of updated databases within one program. It automatically uses the DBID defined in Predict for every access or update statement while the Commit and Close statements will be pointed to the default database and ATM will take care of the synchronization.

Relational Null Support

Adabas supports relational Null fields. The Null field has an indicator in two binary Byte format which indicates whether the field has a value or is Null. This indicator appears in the Adabas record and value buffers.

The definition of a Null field in Predict is shown by 'R' or 'U' in the field Suppression Column.

Adabas Native SQL supports Null fields in the following three clauses:

1. SELECT clause

Every field specified in the SELECT clause which has a Null value indication is generated in the record buffer as two fields. The first field is the Null value indicator as two binary Bytes and its name is the field name, prefixed with "S-". The second field is the field itself.

This definition is generated for every Null field even if it belongs to a group, or even if SELECT* is used.

When the record is read from the database, a value of zero in the Null field indicator means that the value in the field itself is a real value. A value of "-1" ("x'FFFF'") in the Null field indicator means that the field has no value and is a real Null.

2. UPDATE/STORE clauses

There is a new reserved word "NULL" which may be specified as a value for Null fields. For example:

```
SET field=NULL
```

Adabas Native SQL will move "-1" ("x'FFFF'") to the Null field indicator of the specified field in the record buffer used for updating the file.

If the user uses the SET clause and specifies a real value or a variable for a field which has a Null value indicator, Adabas Native SQL will automatically reset the Null field indicator of that field. If the user does not specify the SET clause, but initiates the fields in the record buffer by himself, he should also reset or turn on the Null field indicator.

3. WHERE clause

There is an extension to the syntax:

```
WHERE descriptor IS [NOT] NULL
```

This may be used in order to search for all records where the specified descriptor is Null or not Null. This extension is allowed only for descriptors which are defined with the new relational Null support.

Long Alpha field Support

Adabas has a field format "LA", standing for Long Alpha field.

This format represents a variable field whose length may be up to 16K Bytes.

Because it is a variable field, Adabas returns its value together with two binary Bytes in front of the value which represents the actual length of the field (the length includes the two binary Bytes.).

The definition of a Long Alpha field in Predict uses the format "AV".

Adabas Native SQL generates a Long Alpha field as two separate elements in the record buffer. The first element is the field length as two binary Bytes with the name suffixed with "-LEN". Immediately after is the the second element, which is the definition of the field itself as a character string with a total length taken from Predict with the name suffixed with "-TXT".

Because Adabas returns the value of the field in a variable way, it is impossible to have a definition of a field following the Long Alpha field in the record buffer.

For this reason the following restrictions hold:

- the Long Alpha field may be generated only as the last element in the record buffer.
- Only an elementary field is supported as a Long Alpha field (no MU or PE allowed).

3 SINGLE AND MULTIPLE-RECORD PROCESSING

- Single-Record Processing 40
- Multiple-Record Processing 41

Adabas Native SQL data retrieval statements operate in one of two modes: single-record processing mode and multiple-record processing mode.

The READ ISN statement always operates in single-record mode. The Adabas Native SQL statements in the following list can be used in either single-record processing or multiple-record processing mode:

- COMPARE
- FIND
- FIND COUPLED
- HISTOGRAM
- READ LOGICAL
- READ PHYSICAL SEQUENCE
- SORT.

Adabas Native SQL generates the appropriate data declarations and code for multiple-record processing if the keyword FOR is present in the DECLARE clause of the statement (see list above). If the FOR keyword is not coded or if the DECLARE clause is omitted, Adabas Native SQL generates code for single-record processing.

This chapter covers the following topics:

Single-Record Processing

If single-record processing is to be used, the OPEN, FETCH and CLOSE statements are not required and only the FIND, READ, etc., statement is required. Adabas Native SQL generates executable code from this statement, which must therefore appear in the procedure division of COBOL programs. In FORTRAN, the statement must be included within the executable statements.

Single-record processing should be used if the user needs to access only one record from the file.

Example (single-record processing):

```
.  
. .  
. .  
EXEC ADABAS  
    SELECT PERSON  
    FROM PERSONNEL  
    WHERE PERSONNEL-NUMBER = 180001  
END-EXEC  
DISPLAY NAME FIRST-NAME AGE SEX.
```


In this example, the program uses the single-record processing method to display data from the record located by the WHERE criterion.

Multiple-Record Processing

The OPEN, FETCH and CLOSE statements are used for multiple-record processing. The set of records to be processed is determined using a COMPARE, FIND, HISTOGRAM, READ or SORT statement, followed by an OPEN statement. The records are then processed one by one using the FETCH statement, which will normally be coded in a loop. Finally, the CLOSE statement, which releases the ISN list and other Adabas resources, must be issued if the records were located by a FIND, COMPARE or SORT statement, i.e., if an ISN list was created.

It is the FETCH statement that actually reads each record from the database file and retrieves the values in the fields specified in the SELECT clause of the COMPARE, FIND, HISTOGRAM, READ or SORT statement. The OPEN, FETCH and CLOSE statements generate executable Adabas commands, whereas the COMPARE, FIND, HISTOGRAM, READ or SORT statement merely sets up parameter lists for later use.

The keyword FOR must be specified in the DECLARE clause of COMPARE, FIND, HISTOGRAM, READ or SORT in multiple-record processing mode. Using the DECLARE clause, you define a cursor that associates a 'cursor-name' with the statement. Once the cursor has been defined, it may be referred to in the OPEN, FETCH and CLOSE statements. These statements have the following syntax:

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

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

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

cursor-name is the name used in the FIND, READ, SORT, COMPARE or HISTOGRAM statement that was previously declared. The cursor-name provides the link between the parameter-defining statement (FIND, READ, SORT, COMPARE or HISTOGRAM) and the corresponding executable statements (OPEN, FETCH and CLOSE).

Example (multiple-record processing):

```
.  
. .  
. .  
. .  
EXEC ADABAS  
    DECLARE PERS CURSOR FOR  
    SELECT PERSON  
    FROM PERSONNEL  
    WHERE NAME > = 'BROWN'  
END-EXEC  
. .  
. .  
EXEC ADABAS  
    OPEN PERS  
END-EXEC  
EXEC ADABAS  
    FETCH PERS  
END-EXEC  
PERFORM READ-PERSONNEL UNTIL ADACODE = 3.  
EXEC ADABAS  
    CLOSE PERS  
END-EXEC  
. .  
. .  
READ-PERSONNEL.  
    DISPLAY NAME FIRST-NAME AGE SEX.  
EXEC ADABAS  
    FETCH PERS  
END-EXEC
```

4 OVERVIEW OF STATEMENTS

▪ Syntax	44
▪ Overview of Adabas Native SQL Statements	47
▪ Database Query Statements	47
▪ Data Storage READ Statements	48
▪ Associator READ Statement	49
▪ Statements for Processing Multiple Records	50
▪ Database Modification Statements	50
▪ Logical Transaction Processing Statements	51
▪ Checkpointing Statement	52
▪ Other Adabas Native SQL Statements	52
▪ Adabas Native SQL Clauses	53

This chapter covers the following topics:

Syntax

The Adabas Native SQL statements use the following syntax conventions:

- Upper Case
- Lower Case
- Braces
- Brackets
- Ellipsis
- Ellipsis Preceded by a Comma
- Other Special Characters
- Syntax Diagram for Adabas Native SQL Data Retrieval Statements
- Syntax Diagram for statement-name

Upper Case

Words printed in upper case must be entered exactly as they appear in the definition. However, if the initial part of an upper-case word is underlined, it may be abbreviated by entering only the underlined portion.

Lower Case

Words or hyphenated terms printed in lower-case are either the names of further syntax definitions, or else they are self-descriptive words that must be replaced by a suitable substitution. For example, the first term in the syntax definition shown below is *statement-name*, which is in turn described in the next syntax definition; the word *constant* is self-descriptive and might be replaced by the number 667.

Braces

Braces {} are used:

- to enclose alternatives, which are either stacked vertically, or stacked horizontally and separated by vertical bars. One of the alternatives must be coded. Default values that apply when a parameter is omitted are underlined.
- to group terms together. Ellipsis (see below) following the closing brace applies to the entire group, that is, to everything within the braces.

Brackets

Brackets [] indicate that the enclosed expression is optional.

Ellipsis

Ellipsis (a series of dots ...) after a term indicates that the term may be repeated. If the ellipsis follows a bracketed expression, the whole of the expression must be repeated. Ellipsis followed by a number, for example ...₄, indicates the maximum number of times that the term may be coded.

Example:

A...₃

denotes any one of the following strings:

A
AA
AAA

Ellipsis Preceded by a Comma

Ellipsis preceded by a comma (,...) after a term indicates that the term may be repeated; if it is repeated, the occurrences must be separated by commas. Ellipsis preceded by a comma and followed by a number, for example ,...₃, indicates the maximum number of times that the term may be coded. Example:

X,...₃

denotes any one of the following strings:

X
X, X
X, X, X

Other Special Characters

Other special characters, for example comma, asterisk * or parentheses () must be coded exactly as they appear in the definition.

Syntax Diagram for Adabas Native SQL Data Retrieval Statements

```

EXEC ADABAS
  statement-name
  [ DECLARE cursor-name CURSOR [FOR] ]
  [ SELECT { field-name,... | * } ]
  FROM { file [ alias ] },...
  [ WHERE search-criterion ]
  [ OPTIONS
    [ {
      [ AUTOID
        [ DBID= database-name value1 ]
      } ]
    [ CIPHER= value1 ]
    [ COND-NAME={Y|N} ]
    [ HOLD [RETURN] ]
    [ INDEXED={Y|N} ]
    [ ISN= value2 ]
    [ ISNSIZE= len ]
    [ MAXTIME= value3 ]
    [ PASSWORD= value4 ]
    [ PREFIX= prefix ]
    [ SAVE ]
    [ SEQUENCE ]
    [ STATIC={Y|N} ]
    [ SUFFIX= suffix ]
  ]
  [ ORDER BY def... { DESCENDING
    [ ASCENDING ] } ]
  [ GROUP BY field-name ]
END-EXEC
  
```

Syntax Diagram for statement-name



Overview of Adabas Native SQL Statements

This section describes briefly the function of each Adabas Native SQL statement.

Database Query Statements

Each of these statements produces a list containing the numbers of the database records (ISNs) that satisfy the given retrieval criterion.

If you are only interested in the record whose number appears first in the list, then the database query statement on its own will produce the list and then retrieve the data from this record. However, more generally you will wish to process all of the records identified by the list.

The database query statement, which in this case must include the `DECLARE cursor-name CURSOR FOR` clause, does not retrieve any data. It must be followed by the `OPEN`, `FETCH` and `CLOSE` statements, which are described in section [Statements for Processing Multiple Records](#).

Statement	Action
COMPARE	<p>Produces an ISN list that is a logical combination of two ISN lists that have previously been produced. The ISN list may include all records whose ISNs appear</p> <ul style="list-style-type: none"> ■ in the first list AND in the second list ■ in the first list OR in the second list, or ■ in the first list BUT NOT in the second list. <p>If the keyword FOR is not coded in the DECLARE clause, this statement also reads data from the record whose ISN is at the beginning of this list.</p> <p>This statement generates the Adabas command S8 (Process ISN Lists).</p>

Statement	Action
FIND	<p>Produces an ISN list containing the ISNs of all records that satisfy the given retrieval criterion. If required, the ISN list will be sorted so that the records to which it points can be retrieved in ascending or descending sequence, ordered by the values in one, two or three descriptor fields.</p> <p>If the keyword FOR is not coded in the DECLARE clause, this statement also reads data from the record whose ISN is at the beginning of this list.</p> <p>This statement generates the Adabas command S1/4 (Find Records).</p>
FIND COUPLED	<p>Finds the records in a secondary file that are coupled to a specified record in the primary file. For example, having found a particular record in the PERSONNEL file (primary file), you could use the FIND COUPLED statement to find all the records in the AUTOMOBILES file (secondary file) that detail the cars owned by this employee. The PERSONNEL and AUTOMOBILES files are coupled by the PERSONNEL-NUMBER/OWNER-PERSONNEL-NUMBER fields.</p> <p>If the keyword FOR is not coded in the DECLARE clause, this statement also reads data from the record whose ISN is at the beginning of this list.</p> <p>This statement generates the Adabas command S5 (Find Coupled).</p> <p>Note that this statement is not available under VMS.</p>
SORT	<p>Sorts an ISN list that has been produced by a previous Adabas Native SQL FIND or COMPARE statement. The ISN list is sorted so that the records to which it points can be retrieved in ascending or descending sequence, ordered by the values in one, two or three descriptor fields.</p> <p>If the keyword FOR is not coded in the DECLARE clause, this statement also reads data from the record whose ISN is at the beginning of this list.</p> <p>This statement generates the Adabas command S9 (Sort ISN List).</p>

Data Storage READ Statements

These statements read specified data fields from the database. The READ ISN statement always reads from a single record; the remaining statements can also read data from a single record but they will normally be used in conjunction with the OPEN and FETCH statements to read from a series of records.

Statement	Action
READ ISN	<p>Reads data fields from a single record. The ISN of the record is specified by the program.</p> <p>This statement generates the Adabas command L1/4 (Read Record).</p>
READ LOGICAL	<p>Reads data fields from one or more records. The records are read in logical sequence, based on the ascending order of a given descriptor. The program may optionally specify a starting value for the descriptor. The user may request a Descending option.</p> <p>This statement generates the Adabas command L3/6 (Read Logical Sequence).</p>
READ PHYSICAL SEQUENCE	<p>Reads data fields from one or more records. The records are read in the order in which they are physically stored in the database.</p> <p>This is the most efficient method of reading if an entire file is to be processed and the record sequence is not important.</p> <p>This statement generates the Adabas command L2/5 (Read Physical Sequence).</p>

Associator READ Statement

This statement will normally be used with the `DECLARE cursor-name CURSOR FOR` clause and in conjunction with the `OPEN` and `FETCH` statements in order to retrieve all descriptor values sequentially. The first `FETCH` statement will return the lowest descriptor value (and optionally the number of records that contain this value), the second `FETCH` statement will return the next descriptor value, and so forth.

Statement	Action
HISTOGRAM	<p>Reads from the Adabas Associator but does not read from Data Storage. It returns to the user the values of a specified descriptor in ascending sequence. Optionally, it can also return the number of records that contain each descriptor value. The user may request that the order of the values returned be in descending order.</p> <p>This statement generates the Adabas command L9 (Read Descriptor Values).</p>

Statements for Processing Multiple Records

As mentioned above, some of the Adabas Native SQL statements can be used to process multiple records or descriptor values. This applies to the following:

- statements that produce an ISN list (FIND, FIND COUPLED, SORT and COMPARE)
- statements that initiate sequential reading (READ LOGICAL and READ PHYSICAL SEQUENCE), and
- the HISTOGRAM statement, which initiates reading a sequence of descriptor values.

In each case, the records or descriptor values are actually read by a FETCH statement, which is normally executed in a loop. The FETCH statement is preceded by the statement that initiates processing and by the OPEN statement, both of which are executed once only. When as many records as desired have been processed, the program should issue a CLOSE statement to release the ISN list.

Statement	Action
OPEN	This statement must be issued after the statement that initiates reading and before the sequence of FETCH statements that actually retrieve the data from the database.
FETCH	This is the statement that actually retrieves data from the database. Normally it will be executed in a loop until the end-of-data response code is detected.
CLOSE	Performs housekeeping tasks, such as releasing the ISN list, which is no longer required. This statement must be issued after the FIND, FIND COUPLED, SORT and COMPARE statements. Optionally, it may be issued after a READ LOGICAL, READ PHYSICAL SEQUENCE or HISTOGRAM statement.

Database Modification Statements

These three statements modify the data held in the database. Normally, the DELETE and UPDATE statements will be preceded by other Adabas Native SQL statements that find the required record. This record must be placed in hold status so that other programs cannot interfere until the modification is completed.

All of these statements can be disabled by setting the global parameter MODE NOUPD. This can be useful when testing programs, and also for production programs which should not modify the database in any way.

Statement	Action
DELETE	Deletes a record from the database. This statement generates the Adabas command E1 (Delete Record).
INSERT	Inserts a new record in the database. This statement generates the Adabas command N1/2 (Add Record).
UPDATE	Updates the values held in one or more fields of the specified record. This statement is also used to update fields that were previously empty. This statement generates the Adabas command A1 (Record Update).

Logical Transaction Processing Statements

A logical transaction is defined as the smallest unit of change that, when applied to the database, leaves it in a logically consistent state from the point of view of the application. If processing were to be interrupted when a logical transaction had been only partially applied to the database, there would be a logical inconsistency; this state must be avoided at all costs. Adabas has been designed so that these inconsistent states can never occur if the following three statements are used correctly.

Statement	Action
COMMIT WORK	Marks the end of a logical transaction. This statement generates the Adabas command ET (End Transaction).
ROLLBACK WORK	Cancels all modifications made to the files which the user is accessing during the user's current logical transaction. This statement generates the Adabas command BT (Backout Transaction).
READ USERDATA	The COMMIT WORK, CHECKPOINT and DBCLOSE statements allow the program to store additional data in a special data area. This facility would typically be used to store information about the positions of input files, etc., so that processing can be restarted in the event of a system failure. The READ USERDATA statement is used to recover this information. This statement generates the Adabas command RE (Read ET User Data).

Checkpointing Statement

This statement applies only to programs that update a database in exclusive mode.

Statement	Action
CHECKPOINT	Generates a checkpoint entry in the Adabas checkpoint table. This statement generates the Adabas command C1 (Write a Checkpoint).

Other Adabas Native SQL Statements

Statement	Action
BEGIN	This statement must be included as the first Adabas Native SQL statement in every program, with the possible exception of the COPY and GENERATE statements. In Ada programs, it must be coded in the data declaration part of the program; in COBOL programs it must be coded in the DATA DIVISION; and in FORTRAN programs it must be coded in the DATA DEFINITION area of the program.
CONNECT	Indicates the files to be accessed and the access mode (read-only or read and update). Options are included to specify the processing mode, to specify the password to be used to gain access to password-protected files, and to retrieve user data that were written by a previous program (see also the description of the READ USERDATA statement above). This statement generates the Adabas command OP (Open User Session).
COPY	Permits a file layout generated by Predict as Ada, COBOL, FORTRAN or PL/I code to be copied into the program.
DBCLOSE	Flushes the Adabas buffer, so that database updates are written to the physical storage medium. It can be used if desired after a sequence of logically related transactions. In online applications, however, it should only be used at the end of a user session and not at the end of each TP transaction program. This statement generates the Adabas command CL (Close User Session).
GENERATE	The COPY statement copies a file layout that was generated using information contained in the data dictionary into the program. If it has not already been generated using Predict's facilities, or if the data dictionary information may have been changed since the layout was generated, this statement can be used to generate the file layout from the latest information and copy it into the program in a single step.
HOLD	Places a record in hold status. Other programs cannot interfere with this record so long as it is in hold status. A record must be put in hold status before it can be deleted or updated.

Statement	Action
	<p>See also the HOLD option, which can be used with all Adabas Native SQL data retrieval statements except HISTOGRAM.</p> <p>This statement generates the Adabas command HI (Hold Record).</p> <p>See also the RELEASE ISN statement.</p>
RELEASE	<p>Releases an ISN list that was created by a COMPARE, FIND, FIND COUPLED or SORT statement and retained because the SAVE option was coded. This statement will only be required in exceptional circumstances.</p> <p>This statement generates the Adabas command RC (Release Command ID).</p>
RELEASE ISN	<p>Releases a record from hold status. The converse of the HOLD statement.</p> <p>This statement generates the Adabas command RI (Release Record).</p>
RESTORE	<p>Restores the Adabas Native SQL environment after swapping. Used in conjunction with the SAVE statement in CICS programs running in pseudo-conversational mode and in UTM programs with multi-step transactions. Adabas must be running in get-next mode, that is, you must not specify an ISN buffer (ISNSIZE parameter).</p>
SAVE	<p>Makes the Adabas Native SQL environment available to the user, who should save it in a safe place before swapping takes place. Used in conjunction with the RESTORE statement in CICS programs running in pseudo-conversational mode and in UTM programs with multi-step transactions. Adabas must be running in get-next mode, that is, you must not specify an ISN buffer (ISNSIZE parameter).</p>
TRACE	<p>A debugging aid used to switch trace printing of all executed Adabas Native SQL statements on and off.</p>
WHENEVER	<p>Controls generation of code that tests the response code after execution of Adabas Native SQL statements and, if a non-zero response code occurs, branches to a user-written error handling routine.</p>
WRITE TO LOG	<p>Writes data to the Adabas data protection log. The data can subsequently be read using an Adabas utility program. This statement will only be required in exceptional circumstances.</p> <p>This statement generates the Adabas command C5 (Write User Data To Protection Log).</p>

Adabas Native SQL Clauses

The following clauses are common to the data retrieval statements, i.e., COMPARE, FIND, FIND COUPLED, HISTOGRAM, READ ISN, READ LOGICAL, READ PHYSICAL SEQUENCE and SORT.

- DECLARE Clause
- SELECT Clause
- FROM Clause
- WHERE Clause

- [OPTIONS Clause](#)
- [ORDER BY Clause](#)
- [GROUP BY Clause](#)

DECLARE Clause

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

This clause specifies a *cursor-name* that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the *cursor-name*. The *cursor-name* is used to generate the Adabas command-ID unless the [DYNAMCID](#) option is specified in the OPTIONS parameter.

The *cursor-name* may be up to four characters long and cannot contain special characters such as @, #, \$ and %.



Note: In COBOL programs, all *cursor-names* should be exactly four characters long. Otherwise, some compilers may issue warning messages.

If multiple records are to be processed, the `DECLARE cursor-name CURSOR FOR` construction must be used. The keyword FOR indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same *cursor-name*.

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

SELECT Clause



The diagram shows the SQL keyword **SELECT** followed by a large curly brace. Inside the brace, the text *field-name* is followed by an ellipsis *....*. A small asterisk *** is positioned below the ellipsis, indicating that the asterisk is an optional part of the field list.

The SELECT clause indicates which fields are to be retrieved from the database in the file which is specified in the FROM clause. All types of fields may be selected, with the exception of redefined fields and phonetic descriptors. Fields that are not mentioned in the SELECT clause are not included in the record buffer structure, they are not read from the Adabas file and consequently they cannot be referenced later in the program. The fields may be specified either by their full primary names or by appropriate language-specific synonyms as defined in the data dictionary. See [Synonyms](#) for more information.

If you intend to use language-specific synonyms in SELECT clauses and are running Predict 3.1, invert a new superdescriptor in the FDIC file. This superdescriptor must have the 2-character name SN and consist of the following parent fields:

SYNONYM-NAME (CL).

FILE-NAME (CC).

The message DESCRIPTOR SYNONYM will appear in the Adabas Native SQL MESSAGES. The message SYNONYM will appear whether or not this superdescriptor is inverted.

If the SELECT clause is omitted, then no records are processed, but other functions such as search may be performed.

The field expressions are used by Adabas Native SQL when generating the format buffer and record buffer. The field names generated by Adabas Native SQL for the record buffer are generated from the field-names as defined in the data dictionary, except that language-specific synonyms will be used if they have been defined in the data dictionary. The prefix and suffix are added to the basic field-name, invalid characters may be replaced by the 'validation character', and excess characters may be deleted (truncated) if the name is too long. The field attributes, including format, length, etc., are also taken from the data dictionary. The section [Programming Considerations](#) describes the record buffer structure that Adabas Native SQL generates using the SELECT clause, the FROM clause and the definitions stored in the data dictionary.

The name of the record buffer structure is the 'alias' specified in the FROM clause or, if no alias is specified, the file name specified in the FROM clause.

If an asterisk is specified following the keyword SELECT, all the fields within the userview are read.

Example:

```
SELECT *
FROM FINANCE
```

The structure of the Ada record buffer is as follows:

```
type CREDIT_CARDPERS      is array (INTEGER range <>)
                           of STRING (1..0018);
type CREDIT_LIMITPERS     is array (INTEGER range <>)
                           of STRING (1..0004);
type CURRENT_BALANCEPERS is array (INTEGER range <>)
                           of STRING (1..0004);
type OIL_CREDITPERS       is array (INTEGER range <>)
                           of STRING (1..0007);
type INSURANCE_COMPANYPERS is array (INTEGER range <>,
                                     INTEGER range <>)
                           of STRING (1..0025);
type POLICY_AMOUNTPERS    is array (INTEGER range <>,
                                     INTEGER range <>)
                           of STRING (1..0006);
type ON_VACPERS           is array (INTEGER range <>)
                           of STRING (1..0001);
type RECORD_BUFERS is
  record
    PERSONNEL_NUMBER : STRING          (1..0008);
    CREDIT_CARD       : CREDIT_CARDPERS (1..0002);
    CREDIT_LIMIT      : CREDIT_LIMITPERS (1..0002);
    CURRENT_BALANCE   : CURRENT_BALANCEPERS (1..0002);
    OIL_CREDIT        : OIL_CREDITPERS   (1..0010);
    NET_WORTH         : STRING          (1..0008);
    CREDIT_RATING     : STRING          (1..0002);
    INSURANCE_COMPANY : INSURANCE_COMPANYPERS (1..0003,1..0004);
    POLICY_AMOUNT     : POLICY_AMOUNTPERS (1..0003,1..0004);
    COLLEGE           : STRING          (1..0016);
    ON_VAC            : ON_VACPERS       (1..0005);
    INVESTMENT        : STRING          (1..0015);
    SAVINGS           : STRING          (1..0007);
    BANK              : STRING          (1..0020);
    ISN               : INTEGER;
    QUANTITY          : INTEGER;
    RESPONSE_CODE     : SHORT_INTEGER;
  end record;
FINANCE: RECORD_BUFERS;
```



Note: This example shows a record buffer that was generated from an Adabas Native SQL statement with the cursor-name 'PERS'. The periodic group fields are always generated with STRUCT='N'.

The structure of the COBOL record buffer is as follows:



Note: The level-2 name generated for the record buffer includes the cursor-name, if one was specified. The COBOL example below shows a record buffer that was generated from an Adabas Native SQL statement without a cursor-name.

```

01 FINANCE.
  02 RECORD-BUF-0-1.
    03 PERSONNEL-NUMBER    PIC 9(8).
    03 G-MAJOR-CREDIT.
      04 MAJOR-CREDIT      OCCURS 2.
        05 CREDIT-CARD     PIC X(18).
        05 CREDIT-LIMIT    PIC 9(4).
        05 CURRENT-BALANCE PIC 9(4).
    03 OIL-CREDIT          PIC X(7) OCCURS 10.
    03 NET-WORTH           PIC 9(8).
    03 CREDIT-RATING       PIC 9(2).
    03 G-INSURANCE-POLICY-TYPES.
      04 INSURANCE-POLICY-TYPES OCCURS 3.
        05 INSURANCE-COMPANY PIC X(25) OCCURS 4.
        05 POLICY-AMOUNT     PIC 9(6) OCCURS 4.
    03 COLLEGE             PIC X(16).
    03 G-VACATION.
      04 VACATION          OCCURS 5.
        05 ON-VAC          PIC X(1).
    03 INVESTMENT          PIC X(15).
    03 SAVINGS             PIC 9(7).
    03 BANK                PIC X(20).
  02 ISN                   PIC 9(9) COMP.
  02 QUANTITY             PIC 9(9) COMP.
  02 RESPONSE-CODE        PIC 9(4) COMP.

```

The FORTRAN equivalent is as follows:

```

CHARACTER* 8 PERBER
CHARACTER* 18 CCARD (00002)
CHARACTER* 4 CLIM (00002)
CHARACTER* 4 CBAL (00002)
CHARACTER* 52 MAJDIT
CHARACTER* 8 NETRTH
CHARACTER* 2 CREING
INTEGER* 2 CINPES
CHARACTER* 25 INCOM (00003 , 00004)
CHARACTER* 6 POLUNT(00003 , 00004)
CHARACTER* 372 INSPE
CHARACTER* 16 COLEGE
CHARACTER* 1 ONVAC (00005)
CHARACTER* 5 VACION
CHARACTER* 15 INVENT

```

```
CHARACTER* 7 SAVNGS
CHARACTER* 20 BANK
CHARACTER* 507 FINNCE
```



Notes:

1. The cursor is not shown for FORTRAN.
2. Synonyms are assumed to be defined in the data dictionary as shown in Appendix B and truncation is assumed to occur in the middle of the word. (The maximum length of names is operating-system dependent.)
3. The field FINNCE encompasses all other fields and is the equivalent of the record buffer in COBOL and PL/I.

The structure of the PL/I record buffer is as follows:



Note: The level-2 name generated for the record buffer includes the cursor-name, if one was specified. The PL/I example shows a record buffer that was generated from an Adabas Native SQL statement with the cursor-name 'PERS'.

```
DCL 1 FINANCE,
  2 RECORD_BUFFERS_1 UNAL,
  3 PERSONNEL_NUMBER          PIC '(7)99',
  3 G_MAJOR_CREDIT,
  4 MAJOR_CREDIT              (2),
  5 CREDIT_CARD               CHAR (18),
  5 CREDIT_LIMIT              PIC '(3)99',
  5 CURRENT_BALANCE           PIC '(3)99',
  3 OIL_CREDIT                (10) CHAR (7),
  3 NET_WORTH                 PIC '(7)99',
  3 CREDIT_RATING             PIC '(1)99',
  3 G_INSURANCE_POLICY_TYPES,
  4 INSURANCE_POLICY_TYPES (3),
  5 INSURANCE_COMPANY         (4) CHAR (25),
  5 POLICY_AMOUNT             (4) PIC '(5)99',
  3 COLLEGE                   CHAR (16),
  3 G_VACATION,
  4 VACATION                  (5),
  5 ON_VAC                   CHAR (1),
  3 INVESTMENT                CHAR (15),
  3 SAVINGS                   PIC '(6)99',
  3 BANK                      CHAR (20),
  2 ISN                       FIXED BIN(31),
  2 QUANTITY                  FIXED BIN(31),
  2 RESPONSE_CODE             FIXED BIN(15),
  RECORD_BUFFERS CHAR(585) BASED (ADDR(RECORD_BUFFERS_1));
```

FROM Clause

```
FROM {file [alias ] } ,...
```

The FROM clause specifies the file from which data is to be retrieved. This clause is used together with the SELECT clause to generate the record buffer (Ada, COBOL or PL/I) or the equivalent FORTRAN data structure, and to control the retrieval of information from the database. The fields specified in the SELECT clause refer only to the first file named in the FROM clause; however, the retrieval criterion in the WHERE clause can refer to fields from a maximum of 5 physically-coupled files, or a maximum of 16 soft-coupled files.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used. The alias, which should be unique within the program (including linked modules), is required if two or more Adabas Native SQL statements within the module refer to the same file. It can then be used as a qualifier in subsequent Ada, COBOL or PL/I statements that wish to refer to the fields in the respective record buffers. Note that the alias is not preceded by a comma.

Example:

```
SELECT NAME, CITY
FROM PERSONNEL
```

The record buffer has the name 'PERSONNEL'. You may refer to the variables in the record buffer as:

PERSONNEL.NAME	(Ada)
PERSONNEL.CITY	(Ada)
NAME OF PERSONNEL	(COBOL)
CITY OF PERSONNEL	(COBOL)
NAME	(FORTRAN)
CITY	(FORTRAN)
PERSONNEL.NAME	(PL/I)
PERSONNEL.CITY	(PL/I)

If you use the *alias* option:

```
SELECT NAME  
FROM PERSONNEL PERSON-ALIAS
```



then Adabas Native SQL generates a record buffer structure with the name 'PERSON_ALIAS' (Ada or PL/I) or 'PERSON-ALIAS' (COBOL). You may refer to the variables in the record buffer as:

PERSON_ALIAS.NAME	(Ada)
PERSON_ALIAS.CITY	(Ada)
NAME OF PERSON-ALIAS	(COBOL)
CITY OF PERSON-ALIAS	(COBOL)
NAME	(FORTRAN)
CITY	(FORTRAN)
PERSON_ALIAS.NAME	(PL/I)
PERSON_ALIAS.CITY	(PL/I)

WHERE Clause

```
WHERE search-criterion
```

The *search-criterion* specifies the criterion for selecting the records to be read by the retrieval statement. Since individual statements use the *search-criterion* differently, it is explained for each statement separately. Fields taken from files that are not specified in the FROM clause must be qualified, for example, FILE.FIELD or ALIAS.FIELD.

-  **Note:** (for Ada and FORTRAN users): Packed and unpacked fields are generated as character fields, thus search values must include leading zeros in order to pass numeric values to an alphanumeric field. For example, WHERE PERSONNEL-NUMBER = '00000105'.
-  **Note:** (for Ada users): Character constants (literals) used as search values must be padded with leading spaces.

Special restrictions apply when referring to periodic groups, multiple-value fields and multiple-value fields within periodic groups in WHERE clauses. See the [respective sections on multiple value fields](#) for more information.

OPTIONS Clause

```
OPTIONS [ { AUTODBID  
           DBID= database-name } ]  
        [ CIPHER= value1 ]  
  
        [ COND-NAME= { Y  
                     N } ]  
        [ HOLD [RETURN] ]  
  
        [ INDEXED= { Y  
                  N } ]  
        [ ISN= value2 ]  
        [ ISNSIZE= len ]  
        [ MAXTIME= value3 ]  
        [ PASSWORD= value4 ]  
        [ PREFIX= prefix ]  
        [ SAVE ]  
        [ SEQUENCE ]  
  
        [ STATIC= { Y  
                  N } ]  
        [ SUFFIX= suffix ]
```

 **Note:** Not all options apply to each retrieval statement.

AUTODBID Option

This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, the database specified first will be used.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

This option must be specified when accessing a ciphered file.

The keyword CIPHER is followed by an '=' sign and the cipher key (cipher code), which may be a constant of up to 8 characters or the name of a variable containing the cipher key. If the cipher key is specified as a constant, it will appear in the program listings and its security may be compromised. The use of a variable whose value is read in at run-time is recommended. If the cipher key is specified as the name of a variable, it must be preceded by a colon (':').

Great care should be taken to remember the cipher key used when updating a file. If you update a file and subsequently forget the cipher key, the data can never be recovered from the file correctly.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as Level-88 entries.

The value is taken from one of the following sources:

- Local (higher priority): Use the COND-NAME option for the current COMPARE, FIND, HISTOGRAM, INSERT, READ, SORT or UPDATE statement.
- Global (lower priority): Use the COND-NAME clause of the global **OPTIONS** parameter

This option can only be set if field `With Cond. names` in the Predict Modify COBOL Defaults screen is marked with an "X". See also *Generate COBOL Copy Code* in the *Predict Administration Manual*.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

See the section **HOLD Logic** for more information.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

The value for this option is taken from one of the following sources:

- Local (higher priority): Use the INDEXED option for the current COMPARE, FIND, HISTOGRAM, INSERT, READ, SORT or UPDATE statement.
- Global (lower priority): Use the INDEXED clause of the global **OPTIONS** parameter.

This option can only be set if the field *Indexed by* in the Predict Modify COBOL Defaults screen is marked with an "X". See also *Generate COBOL Copy Code* in the *Predict Administration Manual*.

ISN Option

The ISN option may be used with the READ PHYSICAL SEQUENCE and READ LOGICAL statements. In the READ PHYSICAL SEQUENCE statement, it specifies the ISN of the first record to be read. If a record with this ISN does not exist, the record with the next higher ISN will be read. In the READ LOGICAL statement, the ISN option specifies the ISN of the first record to be read from the set of records that satisfy the WHERE clause.

The parameter that follows the keyword 'ISN', namely *value2*, may be either a constant or the name of a variable that contains the ISN. If *value2* is a variable name, it must be immediately preceded by a colon (:), for example ':NAME'.

ISNSIZE Option

The ISNSIZE parameter defines the maximum number of ISNs that can be stored in the ISN buffer. If the number of records that satisfy the selection criterion exceeds ISNSIZE, the excess ISNs are stored by Adabas and retrieved automatically when required. This process is transparent to the programmer.

If this option is not coded locally, that is, as an option in a COMPARE, FIND, FIND COUPLED or SORT statement, the ISNSIZE defined in the global OPTIONS parameter (see) takes effect. If neither a local nor a global ISNSIZE definition is coded, an ISN buffer is not allocated. This latter mode must be used if the file is protected by the 'security by value' facility, or if the SAVE and RESTORE statements are used in CICS or UTM programs.

A larger value for the ISNSIZE parameter may improve processing speed. See your DBA for further advice about selecting an appropriate value for this option.

MAXTIME Option

This option specifies the time limit for Adabas Sx commands.

Specify either a number or a variable containing a number. The default is defined with the parameter `Maximum time for an Sx command` on the Adabas Native SQL Defaults screen.

See section *OP Command*, paragraph *Additions 4* in the *Adabas Command Reference Manual* for more information..

PASSWORD Option

The keyword `PASSWORD` is followed by an '=' sign and then the password, which may be a constant of up to 8 characters or the name of a variable containing the password.



Note: If the password contains special characters i.e. `@#$$%`, it may not be specified as a constant and a variable should be used.

The use of a variable whose value is read in at run-time is recommended. If the password is specified as the name of a variable, it must be immediately preceded by a colon (':').

Example: `PASSWORD = :VAR`

where `VAR` is the name of a variable containing the password.

This option must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless the password is specified globally in the `CONNECT` statement. In this case, Adabas Native SQL will use this password in all generated Adabas commands unless it is overridden by a password specified in the `PASSWORD` parameter of the `OPTIONS` clause for an individual statement.

PREFIX Option

The prefix is taken from one of the following sources:

- Local (highest priority): Use the `PREFIX` option for the current `COMPARE`, `FIND`, `HISTOGRAM`, `INSERT`, `READ`, `SORT` or `UPDATE` statement.
- Global: Use the `PREFIX` clause of the global `OPTIONS` parameter.
- Predict (lowest priority): The current generation default for the respective language are taken from the data dictionary.

The first two options can only be used if the `Field name prefix` field in the `Predict Modify...Defaults` screen for `Ada`, `COBOL`, `FORTTRAN` or `PL/I` is marked with "X", indicating it may be modified by the user. Otherwise the prefix value defined in the data dictionary cannot be overridden.

SAVE Option

Use this option if you need to retain the entire ISN list. The saved ISN list can be used later in COMPARE, FIND and SORT statements. The saved ISN list is discarded when:

- a further Adabas Native SQL statement that creates another ISN list with the same name (same command-ID) is executed, or
- an Adabas Native SQL 'CLOSE' or 'DBCLOSE' statement is executed, or
- the non-activity time limit or transaction time limit is exceeded.

Under these circumstances, response code 9 is returned when the next Adabas command is attempted.

A CLOSE statement must be executed to release the ISN list after every statement that generates an ISN list (COMPARE, FIND, FIND COUPLED and SORT). If the CLOSE statement is not executed, large amounts of storage will be occupied for the remainder of the Adabas session.

SEQUENCE Option

The SEQUENCE option is used only with the READ ISN statement.

If this option is coded, the record with the specified ISN or the next higher ISN is read. The ISN of the record that was read is returned in the field 'ISN', which is appended to every record buffer (see page). If the file does not contain a record having an ISN higher than the specified ISN, end-of-file is signaled. Therefore, when using this option, the flag ADACODE (Ada, COBOL and PL/I) or SQLCOD (FORTRAN) should be checked for end-of-file status.

If this option is not specified, the record with the specified ISN is read. If the file does not contain a record having the specified ISN, an error is reported (response-code = 113). This causes the program to terminate unless a user-written response code interpretation routine is provided.

See also description of the global parameter [ABORT](#).

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded, all buffers generated by Adabas Native SQL will be defined as static.

The value is taken from one of the following sources:

- Local (higher priority): Use the STATIC option for the current COMPARE, FIND, HISTOGRAM, INSERT, READ, SORT or UPDATE statement.
- Global (lower priority): Use the STATIC clause of the global [OPTIONS](#) parameter.

Note: This option can only be set if the field `Static` in the Predict Modify PL/I Defaults screen is marked with an "X". See also *Generate PL/I Include Code* in the *Predict Administration Manual*.

SUFFIX Option

The suffix is taken from one of the following sources:

- Local (highest priority): Use the SUFFIX option for the current COMPARE, FIND, HISTOGRAM, INSERT, READ, SORT or UPDATE statement.
- Global: Use the SUFFIX clause of the global **OPTIONS** parameter.
- Predict (lowest priority): The current generation default for the respective language is taken from the data dictionary.

The first two options can only be used if the `Field name suffix` field in the Predict Modify...Defaults screen for Ada, COBOL, FORTRAN or PL/I is marked with "X", indicating it may be modified by the user. Otherwise the suffix value defined in the data dictionary cannot be overridden.

ORDER BY Clause

`ORDER BY de1... 3 { DESCENDING }
[ASCENDING] }`

The ORDER BY clause specifies the order in which the records are retrieved. It is used in the FIND, HISTOGRAM, READ LOGICAL and SORT statements.

In the FIND and SORT statements, the ISN list may be sorted on up to three descriptors in ascending or descending sequence. In the READ LOGICAL statement, this clause specifies one descriptor that determines the logical sequence in which the records are to be read.

A descriptor used in an ORDER BY clause may not be a member of a periodic group, nor may it be a phonetic descriptor.

The keyword DESCENDING, which may be abbreviated to DESC, specifies descending sequence, otherwise ascending sequence is assumed as default.

GROUP BY Clause

```
GROUP BY field-name
```

The GROUP BY clause is used only in the HISTOGRAM statement. It specifies the descriptor for which the values are to be retrieved. If the 'WHERE' clause is coded, the field used in the GROUP clause must be the same as the field used in the WHERE clause.

5

ADABAS NATIVE SQL STATEMENTS

▪ The BEGIN Statement	72
▪ The CHECKPOINT Statement	73
▪ The CLOSE Statement	74
▪ The COMMIT WORK Statement	75
▪ The COMPARE Statement	77
▪ The CONNECT Statement	87
▪ The COPY Statement	97
▪ The DBCLOSE Statement	98
▪ The DELETE Statement	101
▪ The FETCH Statement	106
▪ The FIND Statement	107
▪ The FIND COUPLED Statement	125
▪ The GENERATE Statement	135
▪ The HISTOGRAM Statement	136
▪ The HOLD Statement	146
▪ The INSERT Statement	148
▪ The OPEN Statement	157
▪ The READ ISN Statement	158
▪ The READ LOGICAL Statement	167
▪ The READ PHYSICAL SEQUENCE Statement	179
▪ The READ USERDATA Statement	187
▪ The RELEASE Statement	189
▪ The RELEASE ISN Statement	190
▪ The RESTORE Statement	191
▪ The ROLLBACK WORK Statement	192
▪ The SAVE Statement	194
▪ The SORT Statement	196
▪ The TRACE Statement	207
▪ The UPDATE Statement	208
▪ The WHENEVER Statement	220
▪ The WRITE TO LOG Statement	318

This chapter covers the following topics:

The BEGIN Statement

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

This statement must appear in every program that uses Adabas Native SQL statements. The only Adabas Native SQL statements allowed to precede this statement are COPY and GENERATE.

- In Ada programs, the BEGIN statement must be coded in the data declaration part of the program.
- In COBOL programs it must be coded in the DATA DIVISION.
- In FORTRAN programs it must be coded in the DATA DEFINITION area of the program.

Adabas Native SQL generates Adabas control blocks, format buffers, search buffers, value buffers and other miscellaneous information in response to the BEGIN DECLARE SECTION statement.

The CHECKPOINT Statement

```
EXEC ADABAS
  CHECKPOINT USER= value1
END-EXEC
```

The CHECKPOINT statement is used by update programs that issue checkpoints. It is only applicable to programs that run in exclusive file control mode. One option is available:

- USER

USER

For user checkpoints made in exclusive file control mode. An Adabas C1 command is generated.

value1 is a constant of 4 characters identifying the checkpoint code or the name of a variable containing the checkpoint code. If *value1* is a variable name, it must be preceded by a colon (':').

Examples:

```
EXEC ADABAS
  CHECKPOINT USER = 'CK01'
END-EXEC
```

```
EXEC ADABAS
  CHECKPOINT USER = :CURRENT-CKPT
END-EXEC
```

The CLOSE Statement

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

This statement is part of the OPEN/FETCH/CLOSE sequence that is used for processing multiple records. See section 3 for further details.

The CLOSE statement must be used in conjunction with the COMPARE, FIND, FIND COUPLED and SORT statements, that is, with those statements that generate an ISN list. It may be used with the HISTOGRAM, READ LOGICAL and READ PHYSICAL SEQUENCE statements, but its use following these statements is not mandatory.

The CLOSE statement releases various Adabas resources, and it also releases the command-ID from the ISN list table. This makes it impossible to refer to the records after the CLOSE statement has been executed. No more FETCH statements can be executed after the CLOSE has taken place.

This statement generates an Adabas RC command.

The COMMIT WORK Statement

```
EXEC ADABAS
  COMMIT WORK
    [ USERDATA= value ]
END-EXEC
```

The COMMIT WORK statement is used to indicate the end of a logical transaction. It should be issued by ET mode users whenever the program has completed the processing of one logical transaction. Failure to do this may lead to an excessively large hold queue in the Adabas work file and eventually to hold queue overflow.

Should the application program ABEND, the status of the database at the time when the last COMMIT WORK was issued will automatically be restored when Adabas is restarted (autobackout).

An Adabas ET (end-of-transaction) command is generated.

USERDATA Clause

```
USERDATA= value
```

The user may write data to the Adabas system file using the 'USERDATA = value' clause. The data can be retrieved using the CONNECT and READ USERDATA statements. value can be a constant enclosed in apostrophes or the name of a variable containing the user data. If value is a variable name, it must be immediately preceded by a colon (:). See the examples below.

If the USERDATA clause is used, a CONNECT statement with a valid user-ID must have been executed. The user-ID that was specified in the CONNECT statement is associated with the user data, and it must be quoted when recovering the user data with a subsequent CONNECT or READ USERDATA statement.

This facility can be used to record information required when performing a restart, for example the positions of files that are being processed sequentially.

The length of the user data, i.e., the number of characters to be written, must not exceed the limit specified in the USERDATA clause of the global **OPTIONS** parameter.

Examples:

```
EXEC ADABAS
  COMMIT WORK
  USERDATA = :USERVAR
END-EXEC
```

```
EXEC ADABAS
  COMMIT WORK
  USERDATA = 'TEST1234'
END-EXEC
```

The COMPARE Statement

```

EXEC ADABAS
  COMPARE [ ISN [ LISTS ] ]
  [ DECLARE cursor-name CURSOR [ FOR ] ]
  [ SELECT { field-name ,... } ]
  FROM file [ alias ]
  WHERE CURSOR= cursor-name { AND
                                OR
                                AND NOT } CURSOR= cursor-name2

  OPTIONS [ { AUTODBDID
              DBID= database-name } ]
          [ CIPHER= value1 ]
          [ COND-NAME= { Y
                       N } ]
          [ HOLD [ RETURN ] ]
          [ INDEXED= { Y
                    N } ]
          [ ISNSIZE= len ]
          [ PASSWORD= value2 ]
          [ PREFIX= prefix ]
          [ SAVE ]
          [ STATIC= { Y
                   N } ]
          [ SUFFIX= suffix ]

END-EXEC

```

The COMPARE statement performs logical processing on two ISN lists that were previously created using FIND, FIND COUPLED or COMPARE statements with the SAVE option. It can compute the intersection (logical AND) or union (logical OR) of two ISN lists, or the set of ISNs that are in one list but not in the other (logical AND NOT).

The two ISN lists to be compared must relate to the same file, and they must be in ascending ISN sequence. Therefore the ORDER BY option is not permitted in the FIND statement that created the ISN lists to be compared.

The ISN lists to be compared must have been created by Adabas Native SQL statements with the SAVE option. They should be released with the CLOSE statement when they are no longer required.

In general, the COMPARE statement will return a list containing the ISNs of many records.

If more than one record is to be processed, the COMPARE statement must contain the DECLARE *cursor-name* CURSOR FOR clause and it must be followed by an OPEN/FETCH/CLOSE sequence as described in chapter [2004-07-02 tbd]. The fields specified in the SELECT clause are available after execution of the FETCH statement.

If only the record whose ISN is at the head of the resulting ISN list is to be processed, the DECLARE clause may be omitted and the fields specified in the SELECT clause are available after execution of the COMPARE statement. In this case Adabas Native SQL generates executable code for the COMPARE statement, which must therefore appear in the procedure division in COBOL programs.

An Adabas S8 command is generated.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword 'FOR' indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single record is to be processed, the **DECLARE** clause may be omitted.

SELECT Clause

```
SELECT { field-name ,... }  
      *
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If an asterisk is specified following the keyword 'SELECT', all the fields within the userview are read.

See also the [previous discussion on the SELECT clause](#) for more information.

FROM Clause

```
FROM { file [ alias ] } ,...
```

The FROM clause specifies the file from which data is to be retrieved. It is used together with the SELECT clause to generate the record buffer and to control the retrieval of information from the database.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

See also the [previous discussion on the FROM clause](#) for more information.

WHERE Clause

```
WHERE CURSOR= cursor-name { AND  
OR  
AND NOT } CURSOR= cursor-name2
```

The WHERE clause is used to specify the cursor names defined in the FIND or COMPARE statements that created the ISN lists. Both of these statements should have the SAVE option.

The COMPARE statement can be used to perform the following logical operations:

AND The resulting ISNs will contain those ISNs that are present in both ISN lists.

OR The resulting ISNs will contain those ISNs that are present in either the first ISN list or the second ISN list or both.

AND NOT The resulting ISN list will contain those ISNs that are present in the first ISN list (identified by *cursor-name1*) but not present in the second ISN list (identified by *cursor-name2*).

OPTIONS Clause

```

OPTIONS [ { AUTODBID
           [ DBID= database-name ] } ]
        [ CIPHER= value1 ]
        [ COND-NAME= { Y
                     N } ]
        [ HOLD [ RETURN ] ]
        [ INDEXED= { Y
                   N } ]
        [ ISNSIZE= len ]
        [ PASSWORD= value2 ]
        [ PREFIX= prefix ]
        [ SAVE ]
        [ STATIC= { Y
                  N } ]
        [ SUFFIX= suffix ]

```

AUTODBID Option

The AUTODBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

The AUTODBID option and the HOLD option may not be used together. This implies to Adabas Native SQL that you are attempting to update a database other than your default database. Also, AUTODBID and DBID may not be used together.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See also the [previous discussion on the CIPHER option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes as level-88 entries the condition names defined in Predict.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With *Cond. names* in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on the COND-NAME option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

The DBID option and the HOLD option may not be used together. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBDID, AUTODBDID-ALL or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from the Predict additional field attribute 3GL specification, *Indexed by*. If no index name is specified here, the name of the multiple-value field or periodic group is used, prefixed with 'I'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on the INDEXED option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISNSIZE Option

The ISNSIZE parameter defines the maximum number of ISNs that can be stored in the ISN buffer. If the number of records that satisfy the selection criterion exceeds ISNSIZE, the excess ISNs are stored by Adabas and retrieved automatically when required. This process is transparent to the programmer. See also the [previous discussion on the ISNSIZE option](#) for more information.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the password clause of the **CONNECT** statement .

See also the [discussion on security options](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *prefix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on the PREFIX option](#) for more information.

SAVE Option

Use this option if you need to retain the entire ISN list. The saved ISN list can be used later in COMPARE, FIND and SORT statements. The saved ISN list is discarded when:

- A further Adabas Native SQL statement that creates another ISN list with the same name (same command-ID) is executed, or:
- An Adabas Native SQL CLOSE or DBCLOSE statement is executed, or:
- The non-activity time limit or transaction time limit is exceeded.

Under these circumstances, response code 9 is returned when the next Adabas command is attempted.

A CLOSE statement must be executed to release the ISN list after every statement that generates an ISN list (COMPARE, FIND, FIND COUPLED and SORT). If the CLOSE statement is not executed, large amounts of storage will be occupied for the remainder of the Adabas session.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter OPTIONS (see page).

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on the STATIC option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name suffix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on the SUFFIX option](#) for more information.

The CONNECT Statement

```

EXEC ADABAS
CONNECT
  [ userid ]
  [ WITH password ]
  [ ACC= file ,... ]
  [ UPD= file ,... ]
  [ EXU= file ,... ]

  [ AND USERDATA INTO var ]

OPTIONS
  [ NORESTRICTED ]
  [ DBID= var ]
  [ MAXISN= value1 ]
  [ MAXHOLD= value2 ]
  [ MAXCID= value3 ]
  [ MAXTIME= value4 ]
  [ TT= value5 ]
  [ TNA= value6 ]
  [ ACODE= value7 ]
  [ WCODE= value8 ]

END-EXEC

```

The CONNECT statement is used to indicate the beginning of a user session and to list the files that will be used and the modes in which they are to be opened. The CONNECT statement should not be issued by modules called from the main program. If a CONNECT statement is used, it must be in the main program and it must include not only the files used by the main program but also those used by all modules called from the main program. It must be the first executable Adabas Native SQL statement in the program, with the possible exception of COPY and GENERATE statements (compare with the BEGIN statement).

If the CONNECT statement is omitted, the program will run in ET mode. Any files can be read and updated, with only the customary password and cipher restrictions on access.

If the program is to run in exclusive-control mode or if files are to be accessed in access-only mode (all attempts to modify the database will be rejected), then the CONNECT statement must be included.

If the Adabas user session is still active when the CONNECT statement is issued (from a previous program that was not terminated correctly with the DBCLOSE statement), a ROLLBACK WORK will be performed and Response Code 9 is returned.

This statement generates an Adabas OP (open) command.

USERID Clause

userid

This clause specifies the user-ID for the user session. A user-ID must be provided if you intend to store and/or read user data and you require this data to be available during a subsequent user session or Adabas session (see also the CHECKPOINT, COMMIT WORK, DBCLOSE, READ USERDATA and WRITE TO LOG statements). The value provided for the user-ID should be unique for this user (that is, it should not be used by any other user). Response Code 48 will be returned if the user-ID is already in use.

The first character must be an upper-case letter or a digit. *userid* may be a constant of up to 8 characters, or the name of a variable containing the user-ID. If *userid* is a variable name, it must be immediately preceded by a colon (':'), for example ':NAME'.



Note: If *userid* is a constant, it must be enclosed in single quotes.

Password Clause

WITH *password*

You may, if you wish, specify the password only once in your program, in the PASSWORD clause of the CONNECT statement. Adabas Native SQL will pass this password to all generated Adabas commands.

If you also code the PASSWORD option in an Adabas Native SQL statement, the local specification overrides the global specification in the CONNECT statement for that statement only.

password may be a constant of up to 8 characters or the name of a variable containing the password. If *password* is a variable name, it must be preceded by a colon (':') for example ':SECRET'.

ACC Clause

```
ACC= file ,...
```

This is a list of the names of the Adabas files to be accessed by the program in access-only (read-only) mode. Any attempt to update a file opened in access-only mode or to add or delete records will be rejected (response-code=19).

If this clause is present, all files to be processed by the program must be listed in the CONNECT statement. An attempt to read a file that was not specified will cause an error (response code=17).

Example:

```
EXEC ADABAS
  CONNECT ACC = PERSONNEL, AUTOMOBILES, FINANCE
END-EXEC
```

This program uses the files PERSONNEL, AUTOMOBILES and FINANCE in access-only mode.

Adabas Native SQL automatically adds the ABEND file to the ACC list so that the error texts corresponding to non-zero response codes can be retrieved from it as required by the response code interpretation routine. The default is UPD.

UPD and EXU Clauses

```
UPD= file ,...  
EXU= file ,...
```

All files updated by the program should be specified in the CONNECT statement. An attempt to update a file that is not specified in the CONNECT statement will cause an error (response code=17).

There are two types of update programs:

ET-mode: These are programs that can update files in parallel with other programs updating the same files. Programs that run in ET mode must put the required record in hold status before updating or deleting it, and must issue the COMMIT WORK statement at the end of each logical transaction. This mode is used for online update programs.

Exclusive mode: These are programs that have exclusive use of the selected files. During the entire execution time, other programs are prevented from updating these files.

Thus, one or more of the following possibilities may be specified:

- 'UPD =' followed by a list of file names, for programs that run in ET mode. The application program should check the response-code after each Adabas Native SQL statement that generates one or more Adabas commands for the value 9, which would mean that an automatic backout has occurred and the program should restart the transaction from the beginning;
- 'EXU =' followed by a list of file names, for EXCLUSIVE mode;

Further information about exclusive control updating may be found in the *Adabas Command Reference Manual* and the *Adabas DBA Reference Manual*. Consult your DBA before writing programs that run in exclusive file control mode or file cluster mode.

Examples:

```
EXEC ADABAS  
  CONNECT 'MEMUNE'  
    ACC = PERSONNEL UPD = AUTOMOBILES  
END-EXEC
```

The program uses the PERSONNEL file in access-only mode and updates the AUTOMOBILES file in ET-logic mode.

```
EXEC ADABAS
  CONNECT 'MEMUNE'
    UPD = PERSONNEL EXU=PERSONNEL
END-EXEC
```

The program uses the PERSONNEL file in access-only mode and updates the PERSONNEL file in ET-logic mode.

USERDATA Clause

AND USERDATA INTO *var*

This clause enables the user to retrieve the user data stored in the Adabas system file by a CHECKPOINT, COMMIT WORK or DBCLOSE statement.

The last USERDATA record that was stored with a CHECKPOINT, COMMIT WORK or DBCLOSE statement is read into var. var must be preceded by a colon (':'), for example ':NAME'.

This option may only be used if the user specified the same user-ID for the current user session and also for the session during which the USERDATA were stored.

OPTIONS Clause

```
OPTIONS
[ NORESTRICTED ]
[ DBID= var ]
[ MAXISN= value1 ]
[ MAXHOLD= value2 ]
[ MAXCID= value3 ]
[ MAXTIME= value4 ]
[ TT= value5 ]
[ TNA= value6 ]
[ ACODE= value7 ]
[ WCODE= value8 ]
```


 **Note:** Default values for all values except DBID are specified in the Predict Modify Adabas Native SQL Defaults screen.

NORESTRICTED option

If this option is used, the Adabas OPEN command will be generated without the RESTRICTED option, so files which are not specified in CONNECT may be added later to the Adabas user queue element.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

 **Note:** Unless using the AUTODBID-ATM global option, only read or search access is permitted if the DBID option is used; the INSERT, UPDATE and DELETE statements must not be used. One CONNECT statement must be issued for each database to be accessed.

MAXISN Option

This option specifies the maximum number of ISNs resulting from the execution of Sx commands that Adabas can store internally in its ISN table. Increasing the default setting may reduce access to the Adabas work file.

MAXHOLD Option

This option specifies the maximum number of records that the user may have in hold status at any time.

MAXCID Option

This option specifies the maximum number of Command IDs that may be active for the user at the same time.

MAXTIME Option

This option specifies the time limit for the execution of Sx commands.

The programmer should consult with the DBA about the system default values for these parameters before changing them. For further details, see the *Adabas Command Reference Manual*, section *OP Command*, paragraph *Additions 4*.

TT Option

This option may be used to specify a transaction time limit.

TNA Option

This option may be used to specify a non-activity time limit.

ACODE option

This option allows for providing the encoding key for "A" format fields during the user session.

WCODE option

This option allows for providing the encoding key for "W" format fields during the user session.

The COPY Statement

1

```
EXEC ADABAS
  COPY file-name [member-name]
END-EXEC
```

2

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

Adabas Native SQL supports the COPY statement as described in the chapter *The Preprocessor* of the *Predict Administration Manual*. A file layout that has been generated as Ada, COBOL, FORTRAN or PL/I code by Predict can be copied into the program by means of this statement.

The *file-name* must always be specified. It is the name of the file as defined in the data dictionary.

The *member-name* must be specified if more than one file layout has been generated for this file.

The *file-name* and *member-name* can be specified as positional parameters (see [1] above) or as keyword parameters (see [2] above).

The DBCLOSE Statement

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

The DBCLOSE statement is used to terminate a user session. All Adabas resources are released.

This statement may be issued at the end of the main program. It should not be issued by modules called by a main program, nor should it be issued at the end of a TP transaction program unless this coincides with the end of the user session.

USERDATA Clause

```
USERDATA= var
```

The user may store user data in the Adabas system file by including the 'USERDATA = *var*' clause. The user data can be retrieved by a subsequent CONNECT or READ USERDATA statement. *var* is the name of the variable containing the user data. The variable name must be immediately preceded by a colon (':'), for example 'USERDATA = :NAME'. The length of the user data, that is the number of characters to be written, must not exceed the limit specified in the USERDATA clause of the global **OPTIONS** parameter.

This statement generates an Adabas CL (close) command.

Example:

```
EXEC ADABAS
  DBCLOSE
  USERDATA = :USERVAR
END-EXEC
```

OPTIONS Clause

```
OPTIONS  
[ DBID= database-name ]
```

DBID Option

This option may be used if the program has accessed more than one database. The *database-name* must be defined in the data dictionary, including the file or files that have been accessed. One DBCLOSE statement should be issued per database.

The DELETE Statement

```
EXEC ADABAS
DELETE
  [ DECLARE cursor-name CURSOR ]
  FROM file1 [alias]

  WHERE { ISN= value
         CURRENT OF cursor-name1 }

  [ OPTIONS
    [ PASSWORD= value1 ]
    [ CIPHER= value2 ] ]

END-EXEC
```

The DELETE statement deletes a record from the specified file. The record to be deleted must be retrieved by the FIND statement or one of the READ statements before issuing the DELETE statement. The record must be in hold status unless the program is running in EXU mode (see the CONNECT statement). A record can be 'held' either by specifying the 'HOLD' option in the statement that reads it, or by issuing a separate HOLD statement. If the record is not in hold status, it will implicitly be 'held'.

When the logical transaction has been completed, a COMMIT WORK statement should be issued. One of the effects of this statement is to release records that are in hold status.

This statement generates an Adabas E1 command.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

The cursor-name may be up to four characters long. The DECLARE clause will not normally be required, but it may be used if desired to define the Adabas command ID.

Note: This clause should not be used if the WHERE CURRENT OF clause is used.

FROM Clause

```
FROM file1 [alias]
```

file1 is the Adabas file name or view name, as defined in the data dictionary, of the file from which the record is to be deleted. If the same file appears in another statement, an *alias* should be used.

WHERE Clause

```
WHERE { ISN= value  
       CURRENT OF cursor-name1 }
```

The WHERE clause is used to specify the ISN of the record to be deleted.

In order to delete a record whose ISN is explicitly known, the 'WHERE ISN = *value*' option should be used. *value* may be a constant or the name of a variable containing the ISN. If *value* is a variable name, it must be immediately preceded by a colon (':'), for example ':NAME'. The colon must not be coded following the '=' sign if *value* is a numeric constant, for example 'WHERE ISN = 1234'.

The option 'WHERE CURRENT OF *cursor-name1*' should be coded in order to delete a record found by a previous Adabas Native SQL statement. *cursor-name1* is the name of the cursor defined in that statement.

OPTIONS Clause

```
OPTIONS  
  [ PASSWORD= value1 ]  
  [ CIPHER= value2 ]
```

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement. See also the [previous discussion on this option](#) for more information.

CIPHER Option

The cipher key must be specified when accessing a ciphered file.

See also the [previous discussion on this option](#) for more information.

The FETCH Statement

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

This statement is part of the OPEN/FETCH/CLOSE sequence that is used for processing multiple records. See [Multiple-Record Processing](#) for more information.

The FETCH statement reads the data from the file into the record buffer. An OPEN statement must always be issued before the first FETCH statement can be executed when using multiple-record processing. Each successive FETCH statement automatically reads the next record (or delivers the next descriptor value in the case of the HISTOGRAM statement), until all the records have been passed to the user program. When all records have been read, an end-of-file condition is encountered and the flag ADACODE is set to 3.

The FIND Statement

```

EXEC ADABAS
  FIND
  [ DECLARE cursor-name CURSOR [ FOR ] ]
  [ SELECT { field-name ,... } ]
  FROM file [ alias ]
  WHERE search-criterion

  [ OPTIONS [ { AUTODBD
    [ DBID= database-name } ]
    [ CIPHER= value1 ]
    [ COND-NAME= { Y
    N } ]
    [ HOLD [ RETURN ] ]
    [ INDEXED= { Y
    N } ]
    [ ISNSIZE= len ]
    [ MAXTIME= value2 ]
    [ PASSWORD= value3 ]
    [ PREFIX= prefix ]
    [ SAVE ]
    [ STATIC= { Y
    N } ]
    [ SUFFIX= suffix ] ] ]

  [ ORDER BY de1... 3 { DESCENDING
  ASCENDING } ]

END-EXEC

```

The FIND statement performs a retrieval query against a database file, selecting the record or records specified by the search criterion in the WHERE clause. The keyword 'FIND' may optionally be omitted.

This statement returns either a list of the ISNs of the records that satisfy the search criterion, or an 'end-of-file' indication in the variable ADACODE (Ada, COBOL or PL/I) or SQLCOD (FORTRAN), indicating that no records satisfied the search criterion.

In general, the FIND statement will return a list containing the ISNs of many records.

If all of the records found by the FIND statement are to be processed, then the FIND statement must include the 'DECLARE *cursor-name* CURSOR FOR' clause and it must be followed by an OPEN/FETCH/CLOSE sequence as **described previously**. The fields specified in the SELECT clause are available after execution of the FETCH statement.

If only the first of these records is to be processed, then the DECLARE clause may be omitted and the fields specified in the SELECT clause are available after execution of the FIND statement. In this case, ADABAS Native SQL generates executable code for the FIND statement, which must therefore appear in the procedure division in COBOL programs.

The FIND statement can only retrieve data from the first file (main file) named in the FROM-clause, although the search criterion can include descriptor fields taken from up to five physically-coupled or 16 soft-coupled files (except in the case of VMS which does not support coupled files). The coupling relationships must be defined in PREDICT. If data fields are to be retrieved not from the main file but from a coupled file, the FIND COUPLED statement must be used in conjunction with the FIND statement.

The FIND statement causes an ADABAS S1/S4 command to be generated, or an S2 command if the 'ORDER BY' clause is coded.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword 'FOR' indicates to ADABAS Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single record is to be processed, the DECLARE clause may be omitted.

See also the [previous discussion on this clause](#) for more information.

SELECT Clause

```
SELECT { field-name ,... }
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If the SELECT clause is omitted, then no records are processed, but other functions such as search may be performed.

If an asterisk is specified following the keyword 'SELECT', all the fields within the userview are read.

See also the [previous discussion on this clause](#) for more information.

FROM Clause

```
FROM file [ alias ]
```

The FROM clause specifies the file or files that contain the fields used in the search criterion. It is also used, in conjunction with the SELECT clause, to generate the record buffer and to control the retrieval of information from the database.

file is the ADABAS file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

In the FIND statement, up to 5 physically-coupled or 16 soft-coupled files may be specified in the FROM clause. This facility is used if the search criterion includes fields taken from more than one file. Every file containing fields used in the search criterion must be listed in the FROM clause. Data can only be retrieved from the first file (main file) whose name directly follows the keyword 'FROM'.

The second and subsequent files listed in the FIND statement must be physically coupled to the main file. Note that the names of the coupled files are separated by commas, but the alias is not preceded by a comma.

See also the [previous discussion on this clause](#) for more information.



Note: In VMS only one file can be specified in the FROM clause, because coupled files are not supported.

WHERE Clause

```
WHERE search-criterion
```

The WHERE clause identifies the set of records to be selected. Only database fields that are defined as descriptors, subdescriptors, superdescriptors, hyperdescriptors or phonetic descriptors may be used to form the search-criterion. In ADABAS Version 5, non-descriptor fields may be used, if the NONDE indication in the ADABAS Native SQL DDA allows it.

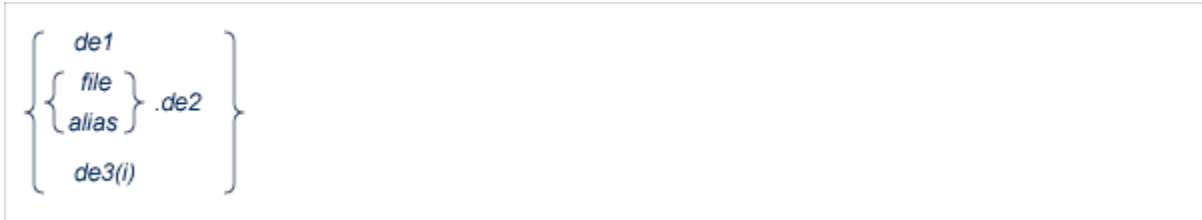
The search-criterion is made up of descriptors, logical operators and values, according to the type of selection relevant to the application.

search-criterion

search-expression [{ **AND**
OR } *search-expression ...*]

search-expression

```
{ { file1 } .de1 = { file2 } .de2  
  { alias1 }  
descriptor comp value  
descriptor BETWEEN value1 AND value2 [exception ]  
descriptor IN ( value,... )  
descriptor IS [ NOT ] NULL  
SETID= 'cursor-name'
```

descriptor

The diagram shows a descriptor structure enclosed in a large right-facing curly brace. Inside this structure, there are four elements: *de1*, a smaller left-facing curly brace containing *file* and *alias*, *.de2*, and *de3(i)*.

comp

NE
EQ
GT
GE
LT
LE
=
>
>=
<
<=

exception

```
AND { filename NOT= value3
      filename NOT BETWEEN value3 AND value4 }
```

de1 is the name of the descriptor to be used in the search expression. The name must refer to a descriptor, subdescriptor, superdescriptor, hyperdescriptor or phonetic descriptor. *de1* is a descriptor in the main file, that is, the file whose name appears first in the FROM clause, directly following the keyword.

The second construct, *file.de2*, shows the name of a descriptor (*de2*) qualified by the filename (*file*). The qualifier is required if the descriptor is in a coupled file, i.e., is not in the main file.

de3 (i) is a reference to a specific occurrence of a descriptor which is a field in a periodic group.

file1.de1 = file2.de2

This construction is used to connect two files via the soft coupling option of Adabas 5. The relationship should exist in Predict.

Example:

```
EXEC ADABAS
  FIND
  SELECT *
  FROM PERSONNEL,AUTOMOBILES
  WHERE NAME = 'SMITH' AND AUTOMOBILES.MAKE = 'FORD'
END-EXEC
```

In this example, NAME is a descriptor field in the main file PERSONNEL, whilst MAKE is a descriptor field in the file AUTOMOBILES which is coupled to the main file.

```
descriptor comp value
```

value is the descriptor value that is to be sought. *value* can be either a constant or the name of a variable. In the latter case, the name must be immediately preceded by a colon (':'), for example ':NAME'.

```
descriptor BETWEEN value1 AND value2 exception
```

The BETWEEN option indicates that any record in which the value of the specified descriptor lies between *value1* and *value2* will satisfy the search expression. *value1* contains the lower limit of the range, and *value2* contains the upper limit of the range.

```
descriptor NOT= value3
```

The NOT = option is used to exclude a specified value of the descriptor from a previous range (given in the BETWEEN option). *value3* must lie between *value1* and *value2* of the preceding BETWEEN option.

```
descriptor NOT BETWEEN value3 AND value4
```

The NOT BETWEEN option is used to exclude a specified range of values from a previous range (given in the BETWEEN option). *value3* and *value4* must lie between *value1* and *value2* of the preceding BETWEEN option.

```
descriptor IN ( value,... )
```

The IN option is used when the user wishes to select records in which a descriptor has any one of a number of values. The search expression is satisfied if the descriptor has any of the values specified in the list.

```
SETID= 'cursor-name'
```

The search expression may also be a *cursor-name* referring to another FIND statement in which the 'SAVE ISN-list' option was used. The search expression denotes the ISN list produced by the previous FIND statement. Records can be selected from this ISN list, so the search can be refined, by combining the SETID option with other search expressions.

```
descriptor IS NULL
```

This search expression will find all records where this descriptor has a relational NULL value (has no value at all).

```
descriptor IS NOT NULL
```

This search expression will find all records where this descriptor has a value.



Note: The order of evaluation of the operators within the Adabas Search Algorithm is:

1. Evaluate the range of values and OR between values of the same descriptor.
2. Evaluate the AND operator.
3. Evaluate the new Logical OR operator (the Logical operator between different descriptors and search criteria).

Examples of Search Criteria

```
AGE = 27
AGE = 27 AND CITY = 'NY'
AGE BETWEEN 25 AND 35
CITY IN ('NY', 'WA', :CITA)
AGE BETWEEN 18 AND 21 OR AGE BETWEEN 65 AND 120
AGE BETWEEN :XAGE AND :YAGE AND AGE > = 18
AGE > 27 AND SETID = 'PERS'
SETID = 'PER1' AND SETID = 'PER2'
AGE BETWEEN 18 AND 30 AND AGE NOT BETWEEN 24 AND 26
AUTOMOBILES.MAKE = 'FORD'
AGE = 30 AND AUTOMOBILES.MAKE = 'FORD'
PERSONNEL.PERSONNEL_NUMBER = AUTOMOBILES.OWNER_PERSONNEL_NUMBER AND ...
```

OPTIONS Clause

```
OPTIONS { { AUTODBID  
          DBID= database-name } }  
        [ CIPHER= value1 ]  
        [ COND-NAME= { Y }  
                   { N } ]  
        [ HOLD [ RETURN ] ]  
        [ INDEXED= { Y }  
                 { N } ]  
        [ ISNSIZE= len ]  
        [ MAXTIME= value2 ]  
        [ PASSWORD= value3 ]  
        [ PREFIX= prefix ]  
        [ SAVE ]  
        [ STATIC= { Y }  
                { N } ]  
        [ SUFFIX= suffix ]
```

AUTODBID Option

The AUTODBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

The AUTODBID option and the HOLD option may not be used together. This implies to Adabas Native SQL that you are attempting to update a database other than your default database. Also, AUTODBID and DBID may not be used together.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See also the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes as level-88 entries the condition names defined in Predict.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With `Cond. names` in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See also the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

The DBID option and the HOLD option may not be used together. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from the Predict additional field attribute 3GL specification, Indexed by. If no index name is specified here, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISNSIZE Option

The ISNSIZE parameter defines the maximum number of ISNs that can be stored in the ISN buffer. If the number of records that satisfy the selection criterion exceeds ISNSIZE, the excess ISNs are stored by Adabas and retrieved automatically when required. This process is transparent to the programmer. See the [previous discussion on this option](#) for more information.

MAXTIME Option

Limits the time of executing the FIND statement. See the [previous discussion on this option](#) for more information.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the password clause of the **CONNECT** statement.

See the [previous discussion on this option](#) for more information

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name prefix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SAVE Option

Use this option if you need to retain the entire ISN list. The saved ISN list can be used later in COMPARE, FIND and SORT statements. The saved ISN list is discarded when:

- A further Adabas Native SQL statement that creates another ISN list with the same name (same command-ID) is executed, or:
- An Adabas Native SQL CLOSE or DBCLOSE statement is executed, or:
- The non-activity time limit or transaction time limit is exceeded.

Under these circumstances, response code 9 is returned when the next Adabas command is attempted.

A CLOSE statement must be executed to release the ISN list after every statement that generates an ISN list (COMPARE, FIND, FIND COUPLED and SORT). If the CLOSE statement is not executed, large amounts of storage will be occupied for the remainder of the Adabas session.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name suffix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

ORDER BY Clause

`ORDER BY de1... 3 { DESCENDING
ASCENDING }`

The ORDER BY clause specifies the order in which the records are retrieved.

The ISN list may be sorted on up to three descriptors in ascending or descending sequence.

A descriptor used in an ORDER BY clause may not be a member of a periodic group, nor may it be a phonetic descriptor.

The keyword DESCENDING, which may be abbreviated to DESC, specifies descending sequence, otherwise ascending sequence as default is assumed. If more than one descriptor is specified, the ASCENDING/DESCENDING option applies collectively to all of them. It is not possible to specify ascending sequence for one descriptor and descending sequence for another.

The FIND COUPLED Statement

```

EXEC ADABAS
  FIND COUPLED
  [ DECLARE cursor-name CURSOR [ FOR ] ]
  [ SELECT { field-name ,... } ]
  FROM file,file2 [ alias1 ]
  WHERE ISN= value

  [ OPTIONS [ { AUTODBDID
               [ DBID= database-name ]
             } ]
           [ CIPHER= value1 ]
           [ COND-NAME= { Y
                         N } ]
           [ HOLD [ RETURN ] ]
           [ INDEXED= { Y
                       N } ]
           [ ISNSIZE= len ]
           [ MAXTIME= value2 ]
           [ PASSWORD= value3 ]
           [ PREFIX= prefix ]
           [ SAVE ]
           [ STATIC= { Y
                      N } ]
           [ SUFFIX= suffix ] ]

END-EXEC

```

The FIND COUPLED statement retrieves fields from a record or records coupled to a given record in another file. Specify the names of both files and the ISN of the record to which the records to be retrieved are coupled.

FIND COUPLED is normally used together with FIND. The FIND statement is used to find a record of interest (the search criterion may include fields from several files); the FIND COUPLED statement is then used to retrieve information from the record or records that are coupled to that record. If more than one record satisfied the search criterion of the original FIND statement, the FIND COUPLED statement must be repeated for each record in the ISN list returned by the FIND statement.

In general, the FIND COUPLED statement will return a list containing the ISNs of several records that are coupled to the specified record in the main file.

If all of the records found by the FIND COUPLED statement are to be processed, then the FIND COUPLED statement must include the 'DECLARE *cursor-name* CURSOR FOR' clause and it must be followed by an OPEN/FETCH/CLOSE sequence as described from page [\[2004-08-24 tbd\]](#). The fields specified in the SELECT clause are available after execution of the FETCH statement.

If, however, only the first of these records is to be processed, then the DECLARE clause may be omitted and the fields specified in the SELECT clause are available after execution of the FIND COUPLED statement. In this case, Adabas Native SQL generates executable code for the FIND COUPLED statement, which must therefore appear in the procedure division in COBOL programs.

Examples including the FIND COUPLED statement may be found in the appendices.



Note: The examples using coupled files cannot be executed under VMS, since coupled files are not supported.

An Adabas S5 command is generated.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword FOR indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single record is to be processed, the DECLARE clause may be omitted.

See the [previous discussion on this clause](#) for more information.

SELECT Clause

```
SELECT { field-name .... }  
      *
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If the SELECT clause is omitted, then no records are processed, but other functions such as search may be performed.

If an asterisk is specified following the keyword SELECT, all the fields within the userview are read.

See the [previous discussion on this clause](#) for more information.

FROM Clause

```
FROM file,file2 [alias1]
```

This is the file list. *file1* and *file2* are Adabas file names or view names as defined in the data dictionary. The two files must be coupled. *file1* is the name of the file from which the records are to be read. *file2* is the name of the file containing the record whose ISN is specified in the WHERE clause.

alias1 is the alias associated with *file1*. If present, it is used as the name of the record buffer; otherwise, the name *file1* is used. The alias - which should be unique within the program (including linked modules) - is required if two or more Adabas Native SQL statements within the module refer to the same file. It can then be used as a qualifier in subsequent Ada, COBOL or PL/I statements that wish to refer to the fields in the respective record buffers.

The names of the coupled files are separated by a comma, but the alias - if present - is not preceded by a comma.

Example:

```
EXEC ADABAS
  FIND COUPLED
  SELECT NAME, CITY
  FROM PERSONNEL,AUTOMOBILES
  WHERE ISN = :VAR
END-EXEC
```

WHERE Clause

```
WHERE ISN= value
```

The WHERE clause specifies the ISN of the record in *file2* to which the records in *file1* are coupled. *value* may be a numeric constant or the name of a variable containing the ISN. If *value* is a variable name, it must be immediately preceded by a colon (':'), for example ':NAME'. The colon must not be coded following the '=' sign if *value* is a numeric constant, for example 'WHERE ISN = 1234'.

OPTIONS Clause

```

OPTIONS [ { AUTODBID
           [ DBID= database-name ] }
        [ CIPHER= value1 ]
        [ COND-NAME= { Y
                     N } ]
        [ HOLD [ RETURN ] ]
        [ INDEXED= { Y
                   N } ]
        [ ISNSIZE= len ]
        [ MAXTIME= value2 ]
        [ PASSWORD= value3 ]
        [ PREFIX= prefix ]
        [ SAVE ]
        [ STATIC= { Y
                  N } ]
        [ SUFFIX= suffix ]

```

AUTODBID Option

The AUTODBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

The cipher key must be specified when accessing a ciphered file.

See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With *Cond. names* in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The database-name must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode. See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

Any specification here will override any setting of the global parameter **OPTIONS**).

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISNSIZE Option

The ISNSIZE parameter defines the maximum number of ISNs that can be stored in the ISN buffer. If the number of records that satisfy the selection criterion exceeds ISNSIZE, the excess ISNs are stored by Adabas and retrieved automatically when required. This process is transparent to the programmer. See the [previous discussion on this option](#) for more information.

MAXTIME Option

This option is used to limit the time of executing the FIND statement. The user may specify a number or variable containing a number.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement. See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *prefix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SAVE Option

Use this option if you need to retain the entire ISN list. The saved ISN list can be used later in COMPARE, FIND and SORT statements. The saved ISN list is discarded when:

- A further Adabas Native SQL statement that creates another ISN list with the same name (same command-ID) is executed, or:
- An Adabas Native SQL CLOSE or DBCLOSE statement is executed, or:
- The non-activity time limit or transaction time limit is exceeded.

Under these circumstances, response code 9 is returned when the next Adabas command is attempted.

A CLOSE statement must be executed to release the ISN list after every statement that generates an ISN list (COMPARE, FIND, FIND COUPLED and SORT). If the CLOSE statement is not executed, large amounts of storage will be occupied for the remainder of the Adabas session.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name suffix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

The GENERATE Statement

```

EXEC ADABAS
  GENERATE
  FILE= file-id

  [
  OPTIONS
  [
  COND-NAME= { Y }
             { N }
  ]
  [
  INDEXED= { Y }
           { N }
  ]
  [ PREFIX= prefix ]
  [ START-LEVEL= start-level ]
  [
  STATIC= { Y }
          { N }
  ]
  [ SUFFIX= suffix ]
  ]

END-EXEC

```

This statement is used to generate copy code you wish to include in your program or to regenerate copy code for which the Predict dictionary definitions have been modified after generation.

This Adabas Native SQL statement provides a subset of the facilities provided by the Predict GENERATE statement. If you require any of the extended range of facilities, use the Predict pre-processor.

If more than one preprocessor is used, they must be used in the following order:

- Predict
- Adabas Native SQL
- CICS.

The *start-level* is in the range 1..40.

See the description of the analogous GENERATE command in chapter *The Preprocessor* of the *Predict Administration Manual* for more information.

The HISTOGRAM Statement

```

EXEC ADABAS
  HISTOGRAM
  [ DECLARE cursor-name CURSOR [ FOR ] ]
  [ SELECT [ field-name [(i)] ] [ COUNT(*) ] ]
  FROM file [ alias ]

  [ WHERE { field-name [(i)] BETWEEN value1 AND value2 }
           { field-name [(i)] { {GE | >=}
                                {GT | >}
                                {LE | <=}
                                {LT | <} } value3 } ]

  [ OPTIONS
    [ { AUTODBID
      [ DBID= database-name ] } ]
    [ COND-NAME= { Y
                  N } ]
    [ PASSWORD= value4 ]
    [ PREFIX= prefix ]
    [ STATIC= { Y
               N } ]
    [ SUFFIX= suffix ] ]

  ORDER BY field-name1 { DESCENDING
                          ASCENDING }

  GROUP BY field-name [(i)]

END-EXEC

```

The HISTOGRAM statement is used to determine the values present for a given descriptor in the specified file. The values are returned in ascending or descending sequence. Along with each descriptor value, Adabas Native SQL indicates the number of records that contain that value. This information is read from the Associator inverted lists; no access is made to Data Storage.

The HISTOGRAM statement will normally be used to read many descriptor values in sequence. In this case, the 'DECLARE *cursor-name* CURSOR FOR' clause must be coded, and the HISTOGRAM statement must be followed by the OPEN and FETCH statements. The descriptor field

specified in the SELECT clause and also the QUANTITY, i.e., the number of records with that descriptor value, are available after execution of the FETCH statement.

If only the first (lowest) descriptor value that is greater than or equal to the specified starting value is required, the DECLARE clause may be omitted. The descriptor field specified in the SELECT clause is available directly after execution of the HISTOGRAM statement.

An Adabas L9 command is generated.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple descriptor values are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword FOR indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single descriptor value is to be processed, the DECLARE clause may be omitted.

See the [previous discussion on this clause](#) for more information.

SELECT Clause

```
SELECT [ field-name [(i)] ] [ COUNT(*) ]
```

field-name is the name of the descriptor for which the values are to be returned. *field-name* must be the same descriptor as in the GROUP BY clause. The values are provided in ascending or descending order. Null values are not returned for descriptors that were defined with the null value suppression option.

Use the COUNT(*) option to find out how many records contain each descriptor value. The count will then be returned in the variable QUANTITY attached to the record buffer. Note that the string 'COUNT(*)' must be written without spaces.

If the descriptor is a field within a periodic group, the field 'ISN' (Ada, COBOL or PL/I unless the global parameter 'ABORT .' is specified) or 'SQLISN' (Ada, COBOL or PL/I if the global parameter 'ABORT .' is specified; also FORTRAN) will contain the number of the occurrence in which the returned value is located.

FROM Clause

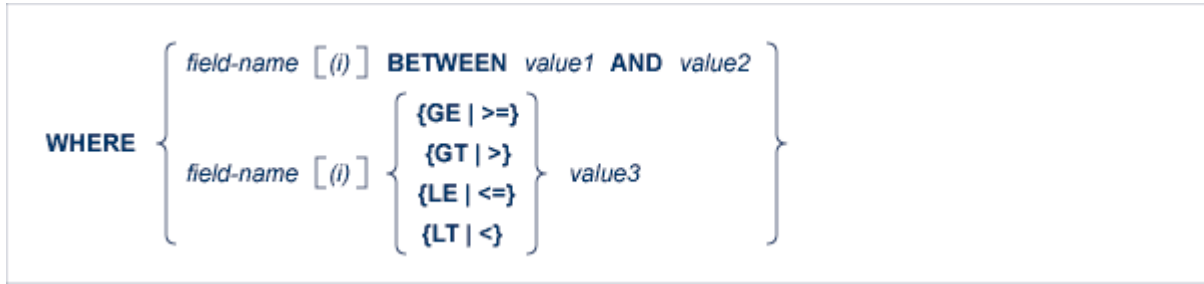
```
FROM file [alias]
```

The FROM clause specifies the file from which the descriptor values are to be retrieved.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

See the [previous discussion on this clause](#) for more information.

WHERE Clause



The range of descriptor values to be read may be restricted by coding an appropriate WHERE clause.

Starting and ending values may be specified using the 'WHERE *field-name* BETWEEN *value1* AND *value2*' option. *value1* represents the value with which reading is to begin and *value2* represents the value with which reading is to end.

The following restriction applies only if ADA-VERSION=62 in the global OPTIONS statement or if the ADA-VERSION= *parameter* is omitted: to specify only a starting value, use the '*field-name* >= *value3*' or '*field-name* GE *value3*' option for ascending order or '*field-name* <= *value3*' or '*field-name* LE *value3*' for descending order (if the Adabas version allows it). *value3* represents the value with which reading is to begin.

In the case of ADA-VERSION=71 in the global OPTIONS statement, all the comparator operators can be used for both ascending and descending order.

The *field-name* must be the descriptor specified in the GROUP BY clause. If the starting value (*value1*, *value3*) is not contained in the file, the next higher value in the list will be used. If no higher value exists, an end-of-file condition will result. *value1*, *value2* and *value3* may be constants or the names of variables containing the values. If they are variable names, they must be immediately preceded by colons (:), for example ':NAME'.

field-name (*i*) is a reference to an occurrence within a periodic group.



Note: If a prefix or suffix is used for a field-name specified in the data dictionary, you may not use the BETWEEN option if ADA-VERSION=62 in the global OPTIONS statement or if the ADA-VERSION= *parameter* is omitted.

OPTIONS Clause

OPTIONS
[{ AUTODBD }]
[DBID= <i>database-name</i>]
[COND-NAME= { Y }]
[COND-NAME= { N }]
[PASSWORD= <i>value4</i>]
[PREFIX= <i>prefix</i>]
[STATIC= { Y }]
[STATIC= { N }]
[SUFFIX= <i>suffix</i>]

AUTODBD Option

The AUTODBD option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in PREDICT as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS**) will be overridden.

With Cond. names in the PREDICT Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the **CONNECT** statement.

PREFIX Option

If the option 'PREFIX = prefix' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS**) or taken from PREDICT.

Field name `prefix` in the PREDICT Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

`Static` in the PREDICT Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from PREDICT.

Field name `suffix` in the PREDICT Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

ORDER BY Clause

```
ORDER BY field-name1 { DESCENDING  
                          ASCENDING }
```

The *field-name1* parameter specifies the descriptor that is to control the reading sequence. Note that the descriptor specified may not be a member of a periodic group, nor may it be a phonetic field.

If the descriptor was defined with the null value suppression option, records having a null value in the descriptor field will not be read.

If the WHERE clause is coded, the descriptor field used in the WHERE clause must be the same as the descriptor field used in the ORDER BY clause.

If DESCENDING is specified, the records are read in descending order.



Note: The 'GE' operator cannot be specified together with the DESCENDING keyword if ADA-VERSION=62 in the global OPTIONS statement or if the ADA-VERSION=*parameter* is omitted.

GROUP BY Clause

```
GROUP BY field-name [i]
```

field-name is the descriptor for which the values are to be returned.

The descriptor may not be a phonetic descriptor or a field in a periodic group. The use of descriptors defined as multiple-value fields is not recommended.

If the SELECT, WHERE and/or ORDER BY clauses are coded, the *field-name* used in these clauses must be the same as the *field-name* used in the GROUP BY clause.

The HOLD Statement

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

This statement is used to place a record in hold status. This reserves the record for subsequent updating or deleting, preventing other users from updating the record until it is released by a COMMIT WORK, RELEASE or ROLLBACK WORK statement. This statement should be used after reading the record and before updating or deleting it, unless the read statement itself included the HOLD option. The *cursor-name* identifies the statement that retrieved the record.

OPTIONS Clause

OPTIONS RETURN

If the RETURN option is coded and the record is already being held for another program, the value 145 will be returned in the response-code. This will cause an error printout followed by an ABEND unless a user-written response code interpretation routine is provided.

See description of the **ABORT** parameter for more information.

If the RETURN option is not coded and the record is being held for another program, the program will be suspended until the record is released.

In many applications, it is preferable to specify the HOLD option in the READ or FIND statement rather than to use a separate HOLD statement.

This statement generates an Adabas HI command.

The INSERT Statement

```
EXEC ADABAS
  INSERT
    INTO file [alias]
    [ DECLARE cursor-name CURSOR ]
    [ WHERE [ ISN= value ] [ CURRENT OF cursor-name1 ] ]
    [ assignments ]

    [ OPTIONS
      [ PASSWORD= value1 ]
      [ CIPHER= value2 ]
      [ PREFIX= value3 ]
      [ SUFFIX= value4 ]
    ]

END-EXEC
```

The INSERT statement adds a new record to the Adabas file.

When an attempt is made to add a new record with one or more fields that have been defined as unique descriptors, response code 198 will be returned if a record having the same descriptor value as the new record already exists in the file. This will cause an error print-out (response code 98 in VAX, otherwise 198) followed by an ABEND unless the user provides an alternative response code interpretation routine. See description of the **ABORT** parameter on page .

This statement generates an Adabas N1 command, or an N2 command if the 'WHERE ISN' clause is coded.

INTO Clause

```
INTO file [alias]
```

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used. The alias, which should be unique, is required if two or more Adabas Native SQL statements within the module refer to the same file. It can then be used as a qualifier in subsequent Ada, COBOL or PL/I statements that refer to the fields in the record buffer.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

This clause specifies a *cursor-name* that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the *cursor-name*. The *cursor-name* may be up to four characters long.

The *cursor-name* specified in the DECLARE clause is used by Adabas as the command-ID. Adabas avoids re-translating the format buffer when it recognizes a command-ID that has been used before, so using this clause can improve performance, particularly if the 'WHERE ISN' option is used.

WHERE Clause

```
WHERE [ ISN= value ] [ CURRENT OF cursor-name1 ]
```

Use one or both options. If both options are used, they can be specified in any order.

The 'WHERE ISN=*value*' option is used if you wish to specify the ISN of the record to be added (user-ISN option). *value* may be either a constant or the name of a variable containing the ISN. The ISN must lie between 1 and the maximum ISN value that was defined for the file. If *value* is a variable name, it must be immediately preceded by a colon (':'), for example ':NAME'. The colon must not be coded following the '=' sign if *value* is a numeric constant, for example 'WHERE ISN = 1234'. If a record with the specified ISN already exists, the value 113 will be returned in the response-code. The 'DECLARE *cursor-name* CURSOR' clause should be used if 'WHERE ISN=*value*' is coded, in order to avoid unnecessary format buffer translations.

If the option 'WHERE CURRENT OF *cursor-name1*' is used and no assignments are used, it is not necessary to build a new record buffer; the existing record buffer is written to the database. This can improve performance.

If the WHERE clause is omitted, the ISN of the new record will be allocated automatically by Adabas.

Assignments

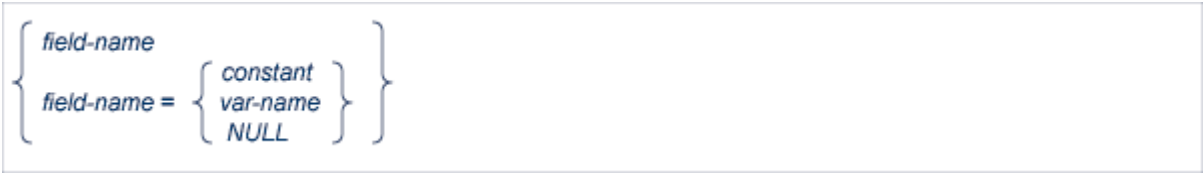


Note: If the option 'WHERE CURRENT OF *cursor-name1*' is used, multiple-value fields and periodic groups are not supported in the assignments. If multiple-value fields or periodic groups are present, all assignments must be made before the statement. No assignments are permitted within the statement.

```
{ SET { expression ... }
  { (field-name ,... ) VALUES ( { constant } ,... ) } }
```

This clause specifies the fields to be written to the record and, optionally, the values to be assigned to them. The expressions may be separated by blanks (spaces) or commas.

A new record buffer is built if this clause is coded. Avoiding this clause may improve performance, because the record buffer of the statement specified in the CURRENT OF clause is used.

expression

field-name denotes the name of the elementary field. This is the full field name as defined in the data dictionary. If necessary, the *field-name* can be subscripted to select the required field from a multiple-value field, from a periodic group, or from a multiple-value field within a periodic group.



Note: *field-name* can be a multiple-value or a part of a periodic group, but in this case an index must be specified within parentheses. For a multiple-value within a periodic group the user should move the value by himself before the INSERT/UPDATE statement.

The option 'SET *field-name*' is used when the required value has already been assigned to the field in the record buffer by means of normal Ada, COBOL, FORTRAN or PL/I statements.

The option 'SET *field-name* = *constant*' or 'SET *field-name* = *var-name*' is used to specify the value to be assigned to the field.

constant denotes a constant (literal) value and *var-name* denotes the name of a variable defined in the Ada, COBOL, FORTRAN or PL/I program, which must be preceded by a colon.

If NULL is specified, Adabas Native SQL will move -1 (x'FFFF') to the Null field indicator of the specified field in the Record buffer used for updating the file.

If the user uses the SET clause and specifies a real value or a variable for a field which has a Null value indicator, Adabas Native SQL will automatically reset the Null field indicator of that field. If the user does not specify the SET clause, but initiates the fields in the Record buffer by himself, he should also reset or turn on the Null field indicator.

var-name

```
{ : var [ (index) ]  
  : root.var [ (index) ]  
  : var [ (index) ] { OF  
                   IN } root }
```

If the variable name is unique within the program, it can be specified as *:var*. Otherwise, it should be made unique by preceding it by *root*, a higher-level data name (qualifier) in the data structure hierarchy. Both the COBOL-type construction (*:var OF root* or *:var IN root*) and the PL/I-type construction (*:root.var*) are valid in Ada, COBOL and PL/I programs.

Both the 'SET *field-name*' option and the 'SET *field-name* = *data*' option can be used in the same SET clause.

The optional *index* may be an integer constant or the name of a variable preceded by a colon. Note that blanks (spaces) are not allowed between the *:var* and the parenthesis preceding the *index*.

OPTIONS Clause

```
OPTIONS  
[ PASSWORD= value1 ]  
[ CIPHER= value2 ]  
[ PREFIX= value3 ]  
[ SUFFIX= value4 ]
```

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the [CONNECT](#) statement.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter [OPTIONS](#) or taken from Predict.

Field name `prefix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter [OPTIONS](#) or taken from Predict.

Field name `suffix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

Example 1: Ada

```

type RECORD_BUFERS is
  record
    SALARY: STRING (...
      .
      .
      .
    end record;
FINANCE: RECORD_BUFERS;

FINANCE.OIL_CREDIT_2:= "0001000";
EXEC ADABAS
  INSERT
    INTO FINANCE
      SET PERSONNEL-NUMBER = "005333756"
      OIL_CREDIT(3) = "0002000"
      OIL_CREDIT(1) = "0001000"
      INSURANCE_COMPANY(1(1)) = "AAA "
      OIL_CREDIT(2)
END-EXEC

```

Example 2: COBOL

```

01 REC
  02 SALARY .....
  02 AGE .....
  02 PERSON-NAME .....hg
  .
  .
  .
MOVE 1000 TO OIL-CREDIT-2
EXEC ADABAS
  INSERT
    INTO FINANCE
      SET PERSONNEL-NUMBER = 5333756
      OIL-CREDIT(3) = 2000
      OIL-CREDIT(1) = 1000
      INSURANCE-COMPANY(1(1)) = 'AAA'
      OIL-CREDIT(2)
END-EXEC

```

Example 3: FORTRAN

```

CHARACTER* 8 PERBER
CHARACTER* 7 OCRE1
CHARACTER* 7 OCRE3
CHARACTER*25 INCOM (00001 , 00001)
CHARACTER* 7 OCRE2
CHARACTER* 14002 FINNCE
.....
OCRE2 = '0001000'

EXEC ADABAS
  INSERT
  INTO FINANCE
  SET PERSONNEL-NUMBER = '005333756'
      OIL-CREDIT(1) = '0002000'
      OIL-CREDIT(3) = '0001000'
      INSURANCE-COMPANY(1(1)) = 'AAA'
      OIL-CREDIT (2)
END-EXEC

```



Note: Synonyms are assumed to be defined in the data dictionary as shown in Appendix B, and truncation is assumed to occur in the middle of the word. (The maximum length of names is operating-system dependent.)

The field FINNCE encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

Example 4: PL/I

```

DCL 01 REC,
     02 SALARY .....,
     02 AGE .....,
     02 PERSON_NAME .....;
.
.
.
OIL_CREDIT_2 = 1000;
EXEC ADABAS
  INSERT
  INTO FINANCE
  SET PERSONNEL-NUMBER = 5333756
      OIL-CREDIT(3) = 2000
      OIL-CREDIT(1) = 1000
      INSURANCE-COMPANY(1(1)) = 'AAA'
      OIL-CREDIT(2)
END-EXEC

```

The OPEN Statement

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

This statement is part of the OPEN/FETCH/CLOSE sequence that is used for processing multiple records.

The OPEN statement processes the parameters defined in the WHERE clause of the statement referenced by *cursor-name* and then issues the actual Adabas command if necessary. Once the OPEN statement has been executed, the contents of the WHERE clause are not re-examined. Therefore, changes to the variables in a WHERE clause will not have any effect until the OPEN statement is re-executed.

In the case of the HISTOGRAM, READ LOGICAL and READ PHYSICAL SEQUENCE statements, the OPEN statement does nothing more than to initialize the variables for the executable Adabas commands. For the COMPARE, FIND, FIND COUPLED and SORT statements, the OPEN statement initializes the variables and also executes the command (FIND, SORT,...) that produces the ISN list. No records are actually fetched from the file until the FETCH statement is executed.

When used in conjunction with a COMPARE, FIND, FIND COUPLED or SORT statement, the OPEN statement puts the ISN quantity in the record buffer. Thus the number of records can be found before executing the first FETCH statement.

The READ ISN Statement

```

EXEC ADABAS
  READ ISN
  [ DECLARE cursor-name CURSOR ]
  [ SELECT { field-name ,... } ]
  FROM file [ alias ]
  WHERE { ISN= value
         CURRENT OF cursor-name1 }

  [ OPTIONS [ { AUTODBD
               [ DBID= database-name ]
             ]
           [ CIPHER= value1 ]
           [ COND-NAME= { Y
                        N } ]
           [ HOLD [ RETURN ] ]
           [ INDEXED= { Y
                      N } ]
           [ PASSWORD= value2 ]
           [ PREFIX= prefix ]
           [ SAVE ]
           [ STATIC= { Y
                    N } ]
           [ SUFFIX= suffix ] ] ]

END-EXEC

```

The READ ISN statement is used to read from a file a single record whose ISN is known, or the first record whose ISN is greater than a specified value.

This statement causes an Adabas L1 command to be generated, or an L4 command if the HOLD option is coded.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long. See the [previous discussion on this clause](#) for more information.

SELECT Clause

```
SELECT { field-name ,... }
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of subdescriptors, superdescriptors and phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If the SELECT clause is omitted, then no records are processed, but other functions such as search may be performed.

If an asterisk (*) is specified following the keyword 'SELECT', all the fields within the userview are read.

See the [previous discussion on this clause](#) for more information.

FROM Clause

```
FROM file [alias]
```

The FROM clause specifies the file from which data are to be retrieved. It is used together with the SELECT clause to generate the record buffer and to control the retrieval of information from the database.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

See the [previous discussion on this clause](#) for more information.

WHERE Clause

```
WHERE { ISN= value  
       CURRENT OF cursor-name1 }
```

The WHERE clause is used to specify the ISN of the record to be read. If the SEQUENCE option is not specified, an error (response-code = 113) will result if the file does not contain a record with this ISN. If the SEQUENCE option is specified and the file does not contain a record with the given ISN, then the record with the next higher ISN will be read, or end-of-file will be signaled if there is no such record.

value can be a constant or the name of a variable. If *value* is a variable name, it must be immediately preceded by a colon (':'), for example ':NAME'. Note that the colon is not part of the '=' sign that follows the 'ISN' keyword.

If the programmer wishes Adabas Native SQL to use the ISN of a record found by a previous statement, he should use the 'CURRENT OF' option, specifying the *cursor-name* of that statement.

OPTIONS Clause

OPTIONS [{ AUTODBID DBID= database-name }] [CIPHER= value1] [COND-NAME= { Y N }] [HOLD [RETURN]] [INDEXED= { Y N }] [PASSWORD= value2] [PREFIX= prefix] [SAVE] [STATIC= { Y N }] [SUFFIX= suffix]

AUTODBID Option

The AUTODBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

The cipher key must be specified when accessing a ciphered file.

See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With Cond. names in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

See [HOLD Logic](#) for more information.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement.

See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name `prefix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SEQUENCE Option

If this option is coded, the record with the specified ISN or the next higher ISN is read. The ISN of the record that was read is returned in the field 'ISN', which is appended to every record buffer. If the file does not contain a record having an ISN higher than the specified ISN, end-of-file is signaled. Therefore, when using this option, the flag ADACODE (Ada, COBOL, PL/I) or SQLCOD (FORTRAN) should be checked for end-of-file status.

If this option is not specified, the record with the specified ISN is read. If the file does not contain a record having the specified ISN, an error is reported (response-code = 113). This causes the program to terminate unless a user-written response code interpretation routine is provided.

See description of the ABORT parameter on page .

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

`Static` in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *suffix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

The READ LOGICAL Statement

EXEC ADABAS**READ LOGICAL**

[**DECLARE** *cursor-name* **CURSOR** [**FOR**]]

[**SELECT** { *field-name* ,... }
 *]

FROM *file* [*alias*]

[**WHERE** *field-name* { **BETWEEN** *value1* **AND** *value2* }
 { { **{GE | >=}** }
 { **{GT | >}** }
 { **{LE | <=}** }
 { **{LT | <}** } } *value3* }]

OPTIONS

[{ **AUTO****ID****BID** }]

[**DBID**= *database-name*]

[**CIPHER**= *value4*]

[**COND-NAME**= { **Y** }
 { **N** }]

[**HOLD** [**RETURN**]]

[**INDEXED**= { **Y** }
 { **N** }]

[**ISN**= *value5*]

[**PASSWORD**= *value6*]

[**PREFIX**= *prefix*]

[**STATIC**= { **Y** }
 { **N** }]

[**SUFFIX**= *suffix*]

ORDER BY *field-name1* { **DESCENDING** }
 { **ASCENDING** }

END-EXEC

Note: The BETWEEN clause only applies if ADA-VERSION=71 in the global OPTIONS statement.

The READ LOGICAL statement is used to read a file, or portion thereof, in logical sequential order based on the ascending or descending sequence of the values for a given descriptor.

This statement will normally be used to read many records in logical sequence. In this case, the 'DECLARE *cursor-name* CURSOR FOR' clause must be coded, and the READ LOGICAL statement must be followed by the **OPEN and FETCH statements**. The fields specified in the SELECT clause are available after execution of the FETCH statement.

If only the first record in the file is required, the DECLARE clause may be omitted and the fields specified in the SELECT clause are available directly after execution of the READ LOGICAL statement.

This statement causes an Adabas L3 command to be generated, or an L6 command if the HOLD option is coded.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword 'FOR' indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single record is to be processed, the DECLARE clause may be omitted.

SELECT Clause

```
SELECT { field-name ,... }  
      *
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of subdescriptors, superdescriptors and phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If the SELECT clause is omitted, then no records are processed, but other functions such as search may be performed.

If an asterisk is specified following the keyword 'SELECT', all the fields in the userview are read.

See the [previous discussion on this clause](#) for more information.

FROM Clause

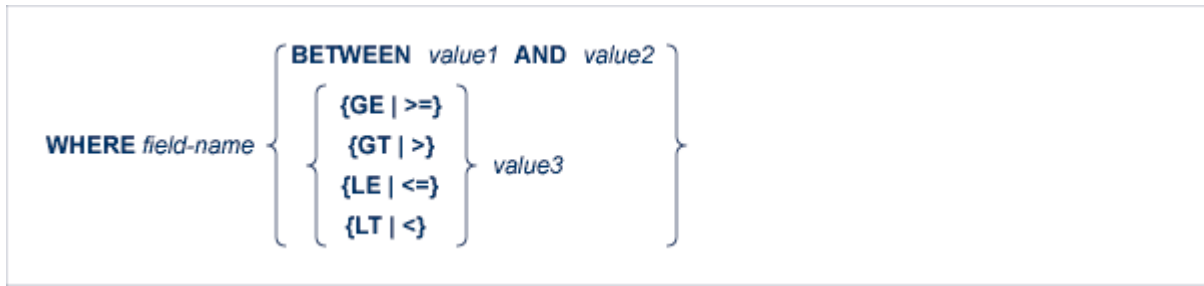
```
FROM file [alias]
```

The FROM clause specifies the file from which data are to be retrieved. It is used together with the SELECT clause to generate the record buffer and to control the retrieval of information from the database.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

See the [previous discussion on this clause](#) for more information.

WHERE Clause



The records may be read starting from a particular descriptor value by using the WHERE clause, where *value* represents the value from which reading is to begin. *field-name1* must be the name of the descriptor specified in the ORDER BY clause (see below).

If the starting value is not found in the file, the next higher value in the file will be used for ascending sequence. If no higher value exists, an end-of-file condition (in ADA, COBOL and PL/I programs: ADACODE = 3; in FORTRAN programs: SQLCOD = 3) will result. *value* may be a constant or the name of a variable.

If *value1*, *value2* or *value3* is a variable name, it must be immediately preceded by a colon (:), for example ':NAME'.

The BETWEEN clause only applies when Adabas Version 7.1 or higher is used and the ADA-VERSION parameter in the global OPTIONS statement is set to 71 or when Adabas Version 3.1 or higher in OpenVMS is used.

OPTIONS Clause

OPTIONS

```
[ { AUTOBID  
  DBID= database-name } ]  
[ CIPHER= value4 ]  
[ COND-NAME= { Y  
              N } ]  
[ HOLD [ RETURN ] ]  
[ INDEXED= { Y  
            N } ]  
[ ISN= value5 ]  
[ PASSWORD= value6 ]  
[ PREFIX= prefix ]  
[ STATIC= { Y  
           N } ]  
[ SUFFIX= suffix ]
```

AUTOBID Option

This option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued. (If the file is linked to more than one database, the DBID option should be used.)

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as Level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With *Cond. names* in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISN Option

The ISN parameter indicates the place within a group of records with the same descriptor value where reading is to begin. Of the records which satisfy the selection criterion *field-name1 = value* (see the WHERE clause), reading begins at the record whose ISN is greater than *value3*. If there is no record with *field-name1 = value* whose ISN is greater than *value3*, the first record with the next descriptor value *field-name1 > value* is read. If there is none, the end-of-file condition (in Ada, COBOL and PL/I programs: ADACODE=3; in FORTRAN programs: SQLCOD=3) will be set.

The ISN value is specified in the *value3* field. *value3* may be a constant or the name of a variable containing the ISN. If *value3* is a variable name, it must be immediately preceded by a colon (:), for example ':NAME'.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement. See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *prefix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name `suffix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

ORDER BY Clause

```
ORDER BY field-name1 { DESCENDING }  
                   { ASCENDING }
```

The *field-name1* parameter specifies the descriptor that is to control the reading sequence. Note that the descriptor specified may not be a member of a periodic group, nor may it be a phonetic field.

If the descriptor was defined with the null value suppression option, records having a null value in the descriptor field will not be read.

If the WHERE clause is coded, the descriptor field used in the WHERE clause must be the same as the descriptor field used in the ORDER BY clause.

If DESCENDING is specified, the records are read in descending order.



Note: If the ADA-VERSION parameter in the global OPTIONS statement is set to 62 or if the ADA-VERSION= parameter is omitted, the 'GE' operator cannot be specified together with the DESCENDING keyword on mainframe platforms, and if the 'LE' operator is specified, the DESCENDING keyword may be omitted on mainframe platforms.

The READ PHYSICAL SEQUENCE Statement

```

EXEC ADABAS
  READ [PHYSICAL [SEQUENCE ]]
  [ DECLARE cursor-name CURSOR [FOR ]]
  [ SELECT { field-name ,... }
        * ]
  FROM file [alias ]

  [ OPTIONS
  [ { AUTODBDID
    [ DBID= database-name } ]
  [ CIPHER= value1 ]
  [ COND-NAME= { Y
                N } ]
  [ HOLD [ RETURN ] ]
  [ INDEXED= { Y
              N } ]
  [ ISN= value2 ]
  [ PASSWORD= value3 ]
  [ PREFIX= prefix ]
  [ STATIC= { Y
             N } ]
  [ SUFFIX= suffix ] ] ]

END-EXEC

```

This statement is used to read records in the sequence in which they are physically located in the data files. It does not read records in any particular logical order.

This statement may be used to read an entire file at maximum speed, since no access is required to the Associator.

This statement is normally used to read many records (possibly the entire file). In this case, the 'DECLARE *cursor-name* CURSOR FOR' clause must be coded, and the READ PHYSICAL SEQUENCE statement must be followed by the **OPEN and FETCH statements**. The fields specified in the SELECT clause are available after execution of the FETCH statement.

If only the first record in the file is required, the DECLARE clause may be omitted and the fields specified in the SELECT clause are available directly after execution of the READ PHYSICAL SEQUENCE statement.

This statement causes an Adabas L2 command to be generated, or an L5 command if the HOLD option is coded.

DECLARE Clause

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

This clause specifies a *cursor-name* that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the *cursor-name*. The *cursor-name* may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword 'FOR' indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same *cursor-name*. If only a single record is to be processed, the DECLARE clause may be omitted.

See the [previous discussion on this clause](#) for more information.

SELECT Clause

```
SELECT { field-name ,... }  
      *
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of subdescriptors, superdescriptors and phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If this clause is omitted, no records are processed, but other functions such as search may be performed.

If an asterisk is specified following the keyword 'SELECT', all the fields within the userview are read.

See the [previous discussion on this clause](#) for more information.

FROM Clause

```
FROM file [alias]
```

This clause specifies the file from which data are to be retrieved. It is used together with the SELECT clause to generate the record buffer and to control the retrieval of information from the database.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used. See the [previous discussion on this clause](#) for more information.

OPTIONS Clause

OPTIONS

```
{  
  AUTOBID  
  DBID= database-name }  
[ CIPHER= value1 ]  
[ COND-NAME= { Y }  
              { N } ]  
[ HOLD [ RETURN ] ]  
[ INDEXED= { Y }  
           { N } ]  
[ ISN= value2 ]  
[ PASSWORD= value3 ]  
[ PREFIX= prefix ]  
[ STATIC= { Y }  
          { N } ]  
[ SUFFIX= suffix ]
```

AUTOBID Option

The AUTOBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With *Cond. names* in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user. A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode.

See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISN Option

This option is used if the file is to be read in physical sequence starting at some position other than the beginning of the file.

The ISN parameter specifies the ISN of the record preceding the record where reading is to begin. The ISN is specified in the *value2* field. *value2* may be a constant or the name of a variable containing the ISN. If *value2* is a variable name, it must be immediately preceded by a colon (:), for example ':NAME'. This field is updated automatically by Adabas and need not be modified by the user every time the next record is to be read.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement. See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name prefix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name suffix in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

The READ USERDATA Statement

```
EXEC ADABAS
  READ USERDATA
    INTO var1
    [ USERID= value ]
END-EXEC
```

This statement reads user data previously stored in the Adabas system file by a CHECKPOINT, COMMIT WORK or DBCLOSE statement.

The user data will be read into the variable whose name is *var1*. The variable name must be immediately preceded by a colon (:), for example 'READ USERDATA INTO :NAME'.

This statement generates an Adabas RE command.

USERID Clause

```
USERID= value
```

If you wish to read data stored by another user, or stored by you during a different Adabas session, the USERID parameter must be used, specifying the user-ID that was used when the data was written. *value* can be an alphanumeric constant or the name of a variable containing the user-ID. If *value* is a variable name, it must be immediately preceded by a colon (:). The colon must not be coded if *value* is a constant.

Examples:

```
EXEC ADABAS
  READ USERDATA INTO :USER-VAR
    USERID = 'US01'
END-EXEC

EXEC ADABAS
  READ USERDATA INTO :TEMP1
    USERID = :HISNAME
END-EXEC
```

The RELEASE Statement

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

You will not normally need this statement. It is used to release the Adabas global format-ID and/or an Adabas command-ID.

The command-ID has three functions:

- to identify a format buffer so that further use of the same format buffer with the same command-ID is more efficient,
- to identify the next READ statement in a sequential read process,
- to identify a list of ISNs found in a FIND statement.

You can release the command-ID from one, two or all three of the above functions. If the FOR clause is not specified, then the command-ID will be released from all the functions and in addition the global format-ID will be released.

Function	Meaning
FORMAT	Releases the command-ID from the internal format buffer pool.
GLOBAL	Releases the ADABAS global format-ID.
SEQ	Releases the command-ID from the table of sequential commands.
LIST	Releases the command-ID from the table of ISN lists.

The command-ID that will be released is the command-ID generated by Adabas Native SQL for the set of buffers identified by *cursor-name*.

If *cursor-name* is not specified, all command-IDs will be released.

See the description of the RC command in the *Adabas Command Reference Manual* for more information.

This statement generates an Adabas RC command.

The RELEASE ISN Statement

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

This statement releases from hold status a record that has been held by a previous READ or HOLD statement with the same *cursor-name* identification.

If you are using ET logic, do not use this statement to release a record that has been updated during your current session.

The COMMIT WORK statement, which is used in ET-mode programs to mark the end of a logical transaction, automatically releases records that were put into hold status during the current transaction.

This statement generates an Adabas RI command.

The RESTORE Statement

```
EXEC ADABAS
  RESTORE
    cursor var1
END-EXEC
```

This statement is used in programs that run under the control of a TP monitor, for example CICS in pseudo-conversational mode or UTM with multiple-step transactions.

The data to be restored must be passed to the RESTORE statement in *var1*, which must have a length of 80 bytes. The name of the variable *var1* must be preceded by a colon (:). The data is passed to the Adabas Native SQL statement identified by *cursor*.

The data must be the same data that was returned from a preceding SAVE statement. The user is responsible for preserving the data between the SAVE statement and the RESTORE statement.

See also the complementary SAVE statement.

The ROLLBACK WORK Statement

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

This statement is used to remove all the database modifications (insertions, deletions and updates) that have been performed since the beginning of the Adabas user session or the last COMMIT WORK or ROLLBACK WORK statement. Note that the ROLLBACK WORK statement can modify the state of files other than the files used in the program that issued the statement. After the ROLLBACK WORK has been completed, the database has the status that it had when the last COMMIT WORK was issued.

This statement generates an Adabas BT (backout transaction) command.

WITHOUT Clause

```
WITHOUT filename
```

The user may backout all files except one by specifying the appropriate file name in the WITHOUT parameter.

Example:

```
EXEC ADABAS  
  ROLLBACK WORK  
    WITHOUT PERSONNEL  
END-EXEC
```

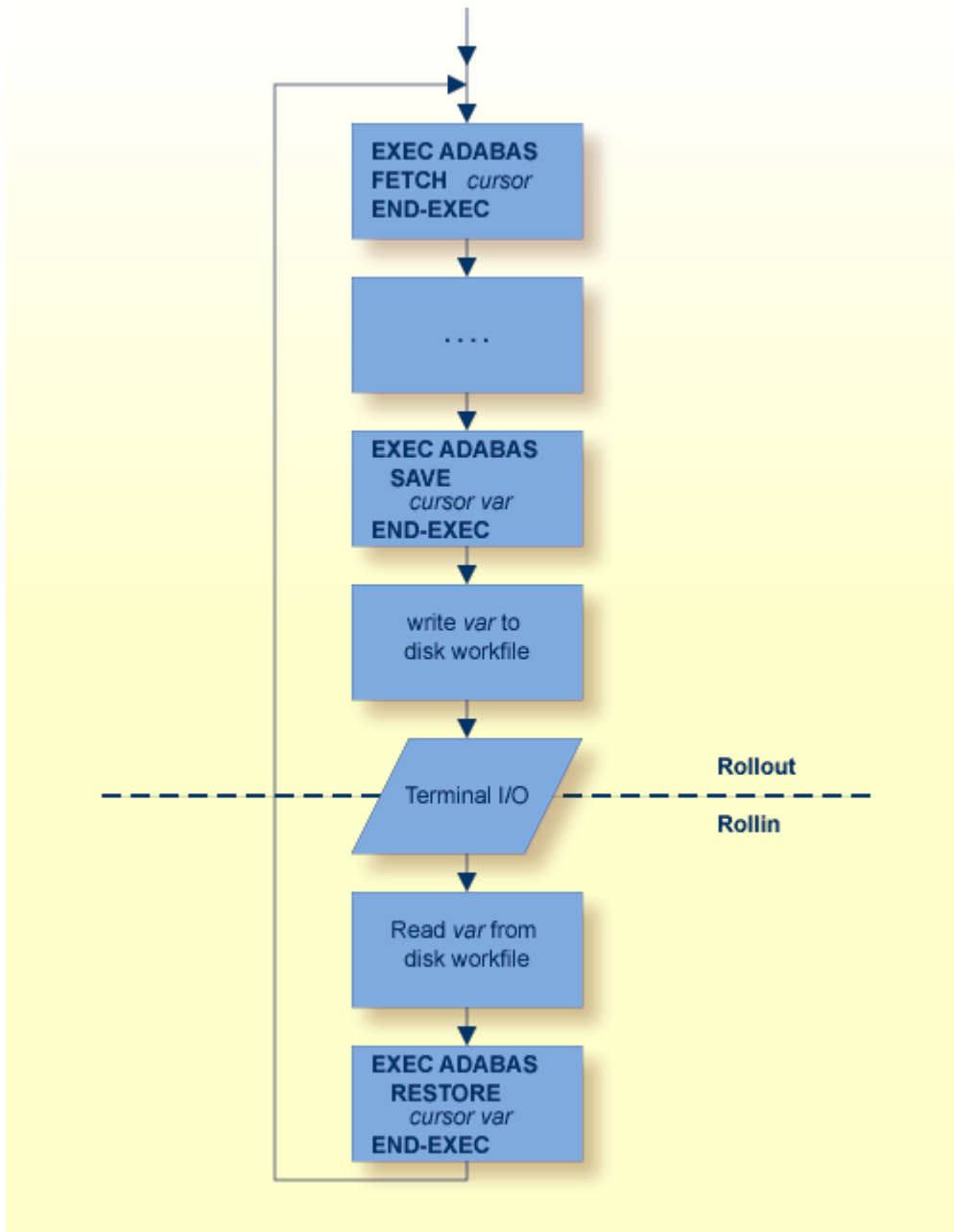
In this example, all files in the database should be backed out with the exception of file PERSONNEL.

The SAVE Statement

```
EXEC ADABAS  
SAVE  
    cursor var1  
END-EXEC
```

This statement is used in programs that run under the control of a TP monitor, for example CICS in pseudo-conversational mode or UTM with multiple-step transactions. Several SAVE statements may be used, one for each Adabas Native SQL statement whose context must be preserved over an I/O transaction. However, unnecessary SAVE statements should be avoided.

A typical sequence of operations is shown in the following diagram:



The data to be saved from the Adabas Native SQL statement identified by cursor is returned in *var1*, which must have a length of 80 bytes. The name of the variable *var1* must be preceded by a colon (:). The data will normally be used in a subsequent RESTORE statement. The user is responsible for preserving the data between the SAVE statement and the RESTORE statement.

See also the complementary RESTORE statement.

Programs that use the SAVE statement must not use the ISNSIZE option in any Adabas SQL statements.

The SORT Statement

```

EXEC ADABAS
  SORT [ ISN [ LISTS ] ]
  [ DECLARE cursor-name CURSOR [ FOR ] ]
  [ SELECT { field-name ,... }
          * ]
  FROM file [ alias ]
  WHERE CURSOR= cursor-name

  [ OPTIONS
  [ { AUTODPID
    [ DBID= database-name } ]
  [ CIPHER= value1 ]
  [ COND-NAME= { Y
                N } ]
  [ HOLD [ RETURN ] ]
  [ INDEXED= { Y
              N } ]
  [ ISNSIZE= len ]
  [ PASSWORD= value2 ]
  [ PREFIX= prefix ]
  [ SAVE ]
  [ STATIC= { Y
             N } ]
  [ SUFFIX= suffix ] ]

  [ ORDER BY de 1..3 { DESCENDING
                     ASCENDING } ]

END-EXEC

```

The SORT statement may be used to sort an ISN list that was created by a COMPARE or FIND statement. The SAVE option must be used in the COMPARE or FIND statement in order to save the ISN list.

The ISN list is sorted according to the values of one, two or three descriptors in the records indicated by the entries in the given ISN list. The keyword DESCENDING, which may be abbreviated to DESC, specifies descending sequence, otherwise ascending sequence will be assumed. If more than one descriptor is specified, the ASCENDING/DESCENDING option applies collectively to

all of them. It is not possible to specify ascending sequence for one descriptor and descending sequence for another.

The ISN list to be sorted must be in ascending ISN sequence. An ISN list that was produced by a FIND statement with the ORDER BY clause or a SORT command cannot be sorted.

In general, the SORT statement will return a list containing the ISNs of many records.

If more than one of the records listed in the ISN list returned by the SORT statement are to be processed, then the SORT statement must include the 'DECLARE *cursor-name* CURSOR FOR' clause and it must be followed by an **OPEN/FETCH/CLOSE sequence**. The fields specified in the SELECT clause are available after execution of the FETCH statement.

If, however, only the first of these records is to be processed, then the DECLARE clause may be omitted and the fields specified in the SELECT clause are available after execution of the SORT statement. In this case, Adabas Native SQL generates executable code for the SORT statement, which must therefore appear in the procedure division in COBOL programs.

An Adabas S9 command is generated.

DECLARE Clause

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

This clause specifies a cursor-name that identifies, or labels, the current statement. Subsequent statements can refer back to a statement that is labeled with a DECLARE clause by quoting the cursor-name. The cursor-name may be up to four characters long.

If multiple records are to be processed, the 'DECLARE *cursor-name* CURSOR FOR' construction must be used. The keyword 'FOR' indicates to Adabas Native SQL that the statement is used in conjunction with OPEN and FETCH statements that appear later in the program quoting the same cursor-name. If only a single record is to be processed, the DECLARE clause may be omitted.

SELECT Clause

```
SELECT { field-name ,... }  
      *
```

The SELECT clause specifies which fields are to be retrieved from the database. All types of fields may be selected, with the exception of subdescriptors, superdescriptors and phonetic descriptors. The fields must be specified by their full names as defined in the data dictionary.

If an asterisk is specified following the keyword 'SELECT', all the fields within the userview are read.

See page for more information.

FROM Clause

```
FROM file [alias]
```

The FROM clause specifies the file from which data are to be retrieved. It is used together with the SELECT clause to generate the record buffer and to control the retrieval of information from the database.

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used.

See the [previous discussion on this clause](#) for more information

WHERE Clause

```
WHERE CURSOR= cursor-name
```

The *cursor-name* is the name coded in the DECLARE clause of the statement that created the ISN list to be sorted. This statement must include the SAVE option. It must not be a FIND statement with the ORDER BY clause or a SORT statement.

OPTIONS Clause

OPTIONS
[{ AUTODBID DBID= <i>database-name</i> }]
[CIPHER= <i>value1</i>]
[COND-NAME= { Y N }]
[HOLD [RETURN]]
[INDEXED= { Y N }]
[ISNSIZE= <i>len</i>]
[PASSWORD= <i>value3</i>]
[PREFIX= <i>prefix</i>]
[SAVE]
[STATIC= { Y N }]
[SUFFIX= <i>suffix</i>]

AUTODBID Option

The AUTODBID option can be used if the file is linked to a single database. This option indicates to Adabas Native SQL that the database ID is to be taken from the data dictionary. If the file is linked to more than one database, an error message will be issued.

If the file is linked to more than one database, the DBID option should be used.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With *Cond. names* in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

DBID Option

This option should be used if the program accesses more than one database. The *database-name* must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed.

This option may not be used together with the HOLD option. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

HOLD Option

If the HOLD option is coded, the record retrieved is placed in hold status. As long as a record is in hold status, it cannot be updated or deleted by any other user.

A record that is to be updated or deleted must be in hold status unless the program is running in exclusive-control mode. See [HOLD Logic](#) for more information.

The HOLD option may not be used together with the AUTODBID or DBID options. This implies to Adabas Native SQL that you are attempting to update a database other than your default database.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

ISNSIZE Option

The ISNSIZE parameter defines the maximum number of ISNs that can be stored in the ISN buffer. For the SORT statement, the ISN buffer must either be defined with size 0, or else it must be large enough to contain the entire ISN list that is to be sorted. If an ISN buffer is defined that is too small to contain the entire ISN list, the value 1 will be returned in the response-code.

The value of len must be either 0 or at least four times the number of records to be sorted.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement (see the [previous discussion on this option](#) for more information).

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *prefix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SAVE Option

The SAVE option is used if the programmer needs to retain the entire ISN list. The saved ISN list is discarded when:

- a further Adabas Native SQL statement that creates another ISN list with the same name (same command ID) is executed, or
- an Adabas Native SQL CLOSE or DBCLOSE statement is executed, or
- the non-activity time limit or transaction time limit is exceeded. Under these circumstances, response code 9 is returned when the next Adabas command is attempted.

A CLOSE statement must be executed to release the ISN list after every statement that generates an ISN list (COMPARE, FIND, FIND COUPLED and SORT). If the CLOSE statement is not executed, large amounts of storage will be occupied for the remainder of the Adabas session.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

`Static` in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name `suffix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

ORDER BY Clause

```
ORDER BY de 1..3 { DESCENDING }  
                  { ASCENDING }
```

The ORDER BY clause specifies the order in which the records are retrieved.

The ISN list may be sorted on up to three descriptors in ascending or descending sequence.

A descriptor used in an ORDER BY clause may not be a member of a periodic group, nor may it be a phonetic descriptor.

The keyword DESCENDING, which may be abbreviated to DESC, specifies descending sequence, otherwise ascending sequence is assumed. If more than one descriptor is specified, the ASCENDING/DESCENDING option applies collectively to all of them. It is not possible to specify ascending sequence for one descriptor and descending sequence for another.

If the ISN list is too big to be sorted, an error is reported with response-code=1. See also the LS (sort work space) parameter in the *Adabas Operations Manual*.

Note: Sorting large ISN lists may take a long time.

The TRACE Statement

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

This statement is used in conjunction with the global option **MODE TRACE**.

Provided 'MODE TRACE.' has been specified, the TRACE ON and TRACE OFF statements can be used within the application program to control trace output statically. Trace output will only be produced in those program sections where TRACE ON is in effect.

Tracing is also controlled dynamically by a variable with the name TRCE (Ada, COBOL, PL/I) or SQDE00 (FORTRAN) in sections where TRACE ON is in effect. The application program can disable tracing dynamically by setting the content of this variable to the value 'OFF', and can re-enable tracing by setting its content to any other value.

Three conditions must be satisfied for tracing to be active:

- the global option 'MODE TRACE.' must be set,
- the 'TRACE ON' statement must be issued, and
- the variable 'TRCE' or 'SQDE00' must not contain the value 'OFF'.



Note: tracing is switched off when the program is started. No trace output will be produced until a TRACE ON statement is executed.

The UPDATE Statement

```

EXEC ADABAS
  UPDATE file [alias ]
  [ DECLARE cursor-name CURSOR ]
  WHERE { ISN= value
         CURRENT OF cursor-name1 }
  [ SET expression ,... ]

  [ OPTIONS
    [ CIPHER= value1 ]
    [ COND-NAME= { Y
                  N } ]
    [ INDEXED= { Y
                 N } ]
    [ PASSWORD= value2 ]
    [ PREFIX= prefix ]
    [ STATIC= { Y
                N } ]
    [ STATUS ]
    [ SUFFIX= suffix ]
  ]

END-EXEC

```

The UPDATE statement is used to update one or more fields of a record in the specified file. The record to be updated must be retrieved by the FIND statement or one of the READ statements before issuing the UPDATE statement. The record must be in hold status unless the program is running in EXU mode (see the CONNECT statement). A record can be 'held' either by specifying the HOLD option in the statement that reads it, or by issuing a separate HOLD statement.

When the logical transaction has been completed, a COMMIT WORK statement should be issued. One of the effects of this statement is to release records that are in hold status.


```
UPDATE file [alias ]
```

file is the Adabas file name or view name as defined in the data dictionary. The *alias*, if present, is used as the name of the record buffer; otherwise, the name *file* is used. The alias, which should be unique, is required if two or more Adabas Native SQL statements within the module refer to the same file. It can be used as a qualifier in subsequent Ada, COBOL, FORTRAN or PL/I statements that refer to the fields in the record buffer.

This statement generates an Adabas A1 command.

DECLARE Clause

```
DECLARE cursor-name CURSOR
```

The cursor-name may be up to four characters long. A cursor-name should be specified if this Adabas Native SQL statement is executed repeatedly; Adabas can recognize the cursor-name, which is also used as the Adabas command-ID, and avoid re-translating the format buffer when the statement is executed subsequently.

WHERE Clause

```
WHERE { ISN= value
        CURRENT OF cursor-name1 }
```

The WHERE clause is used to specify the ISN of the record to be updated.

To update a record having a specific ISN, the programmer should use the 'ISN = value' option. value may be a constant or the name of a variable containing the ISN. If value is a variable name, it must be immediately preceded by a colon (:), for example ':NAME'. The colon must not be coded following the '=' sign if value is a numeric constant, for example 'WHERE ISN = 1234'. If the 'WHERE ISN = value' option is used, the SET clause must be coded.

To update a record using the ISN returned by a previous Adabas Native SQL statement, the programmer should use the 'CURRENT OF' option. *cursor-name1* is the cursor-name defined in that statement.

If the user uses the 'CURRENT OF *cursor-name1*' option in the WHERE clause and the DECLARE and SET clauses are omitted, Adabas Native SQL will use the Adabas variables generated for the statement identified by *cursor-name1* and will not generate variables for this statement. In this case, modify the desired fields before issuing the UPDATE statement.

Example:

```
EXEC ADABAS
  FIND
  DECLARE PERS CURSOR
  SELECT SALARY
  FROM PERSONNEL
  WHERE PERSONNEL-NUMBER = 180001
  OPTIONS HOLD
END-EXEC
.
.
.
SALARY = SALARY * 1.2
EXEC ADABAS
  UPDATE PERSONNEL
  WHERE CURRENT OF PERS
END-EXEC
```

SET Clause

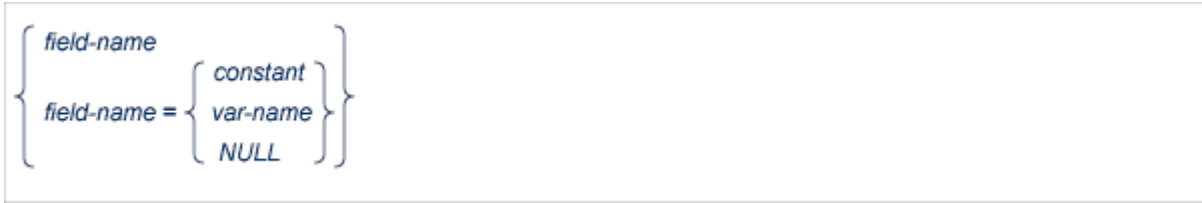
```
SET expression ,...
```

The SET clause specifies the fields to be updated and, optionally, the values to be given to these fields. The expressions may be separated by blanks (spaces) or commas.

The SET clause must always be coded if the option 'WHERE ISN = value' is used.


If the SET clause is coded, it is recommended that the 'DECLARE *cursor-name* CURSOR' clause be used as well to enhance performance.

Coding the SET clause causes Adabas Native SQL to generate a record buffer for this statement. If the SET clause is not coded, the record buffer of the statement referenced by *cursor-name1* will be used to update the database.

expression

field-name denotes the name of the field to be updated. This is the full field name as defined in the data dictionary. If necessary, the *field-name* can be subscripted to select the required field from a multiple-value field, from a periodic group, or from a multiple-value field within a periodic group.

The option 'SET *field-name*' is used when the required value has already been assigned to the field by means of normal Ada, COBOL, FORTRAN or PL/I statements.

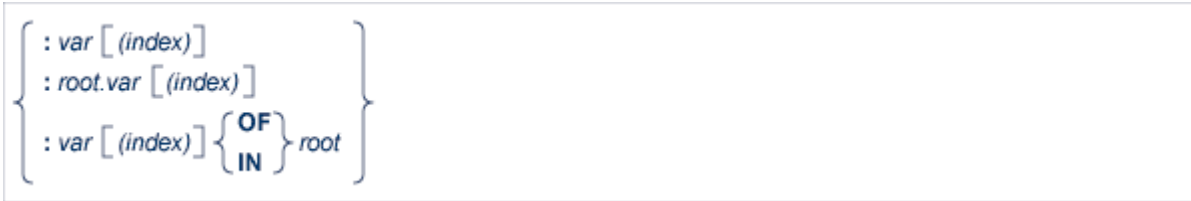
 **Note:** *field-name* can be a multiple-value or a part of a periodic group, but in this case an index must be specified within parentheses. For a multiple-value within a periodic group the user should move the value by himself before the INSERT/UPDATE statement.

The option 'SET *field-name* = *constant*' or 'SET *field-name* = *var-name*' is used to specify the new value to be assigned to the field.

constant denotes a constant (literal) value and *var-name* denotes the name of a variable defined in the Ada, COBOL, FORTRAN or PL/I program, which must be preceded by a colon.

If NULL is specified, Adabas Native SQL will move -1 (x'FFFF') to the Null field indicator of the specified field in the Record buffer used for updating the file.

If the user uses the SET clause and specifies a real value or a variable for a field which has a Null value indicator, Adabas Native SQL will automatically reset the Null field indicator of that field. If the user does not specify the SET clause, but initiates the fields in the Record buffer by himself, he should also reset or turn on the Null field indicator.

var-name

If the variable name is unique within the program, it can be specified as `:var`. Otherwise, it should be made unique by preceding it by `root`, a higher-level data name (qualifier) in the data structure hierarchy. Both the COBOL-type construction (`:var OF root` or `:var IN root`) and the PL/I-type construction (`:root.var`) are valid in Ada, COBOL, FORTRAN and PL/I programs.

Both the 'SET *field-name*' option and the 'SET *field-name* = *data*' option can be used in the same SET clause.

The optional *index* may be an integer constant or the name of a variable preceded by a colon. Note that blanks (spaces) are not allowed between the `:var` and the parenthesis preceding the *index*.

Example 1: Ada

```

type REC_1 is
  record
    SALARY      : STRING (1..6);
    AGE         : STRING (1..2);
    PERSON_NAME: STRING (1..20);
  end record;
REC: REC_1;
.
.
.
EXEC ADABAS
  FIND
  DECLARE PERS CURSOR
  FROM PERSONNEL PRSNNL
  WHERE PERSONNEL_NUMBER = "00180001"
  OPTIONS HOLD
END-EXEC
.
.
PERSONNEL.PHONE_NR = "00746127";
EXEC ADABAS
  UPDATE PERSONNEL
  WHERE CURRENT OF PERS
  SET NAME           = :REC.PERSON-NAME
  AGE               = :REC.AGE
  SALARY            = :REC.SALARY
  PHONE_NR

```

```

        ZIP          = "06100"
        STATE       = "BS"
END-EXEC

```

Example 2: COBOL

```

01 REC
  02 SALARY .....
  02 AGE .....
  02 PERSON-NAME .....
  .
  .
EXEC ADABAS
  FIND
  DECLARE PERS CURSOR
  FROM PERSONNEL PRSNL
  WHERE PERSONNEL-NUMBER = 180001
  OPTIONS HOLD
END-EXEC
  .
  .
MOVE 746127 to PHONE-NR OF PERSONNEL
EXEC ADABAS
  UPDATE PERSONNEL
  WHERE CURRENT OF PERS
  SET NAME   = :PERSON-NAME
    AGE     = :AGE OF REC
    SALARY  = :REC.SALARY
    PHONE-NR
    ZIP     = 35
    STATE  = 'BS'
END-EXEC

```

Example 3: FORTRAN

```

CHARACTER* 20 VARNAM
CHARACTER*  2 VARAGE
CHARACTER*  6 VARSAL
.....
CHARACTER* 20 NAME
CHARACTER*  2 AGE
CHARACTER*  6 SALARY
CHARACTER*  8 PHONE
CHARACTER*  5 ZIP
CHARACTER*  2 STATE
CHARACTER* 43 PERNEL

EXEC ADABAS
  DECLARE PERS CURSOR

```

```

FROM PERSONNEL
WHERE PERSONNEL-NUMBER = '00180001'
OPTIONS HOLD PREFIX=A
END-EXEC

PNONE = '00746127'

EXEC ADABAS
UPDATE PERSONNEL
WHERE CURRENT OF PERS
SET NAME      = :VARNAM
   AGE       = :VARAGE
   SALARY    = :VARSAL
   PHONE
   ZIP      = '35'
   STATE    = 'BS'
END-EXEC

```



Note: Synonyms are assumed to be defined in the data dictionary as shown in Appendix B, and truncation is assumed to occur in the middle of the word. (The maximum length of names is operating-system dependent.)



Note: The field PERNEL encompasses all other fields and is the equivalent of the record buffer in Ada, COBOL and PL/I.

Example 4: PL/I

```

DCL 01 REC,
    02 SALARY ..... ,
    02 AGE ..... ,
    02 PERSON_NAME ..... ;
.
.
EXEC ADABAS
FIND
DECLARE PERS CURSOR
FROM PERSONNEL PRSNL
WHERE PERSONNEL-NUMBER = 180001
OPTIONS HOLD
END-EXEC
.
.
PERSONNEL.PHONE_NR = 746127;
EXEC ADABAS
UPDATE PERSONNEL
WHERE CURRENT OF PERS
SET NAME           = :PERSON-NAME
   AGE            = :AGE OF REC
   SALARY         = :REC.SALARY
   PHONE-NR

```



```

        ZIP           = 6100
        STATE        = 'BS'
END-EXEC

```

OPTIONS Clause

```

OPTIONS
[ CIPHER= value1 ]
[ COND-NAME= { Y }
              { N } ]
[ INDEXED= { Y }
            { N } ]
[ PASSWORD= value2 ]
[ PREFIX= prefix ]
[ STATIC= { Y }
           { N } ]
[ STATUS ]
[ SUFFIX= suffix ]

```

CIPHER Option

The cipher key must be specified when accessing a ciphered file. See the [previous discussion on this option](#) for more information.

COND-NAME Option

This option applies only to COBOL programs.

If the option 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

If specified here, any value specified with the global parameter **OPTIONS** will be overridden.

With Cond. names in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

INDEXED Option

This option applies only to COBOL programs.

If the INDEXED option is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I-'.

Any specification here will override any setting of the global parameter **OPTIONS**.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

PASSWORD Option

The password must be specified in each Adabas Native SQL statement that accesses a password-protected file or a file that is protected by security by value, unless it is specified globally in the CONNECT statement. See the [previous discussion on this option](#) for more information.

PREFIX Option

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *prefix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

STATIC Option

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded here, all buffers generated by Adabas Native SQL will be defined as static. This will override any setting of the global parameter **OPTIONS**.

Static in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

STATUS Option (available with Adabas Version 4 only)

The STATUS option invokes the Status Protection option of Adabas. This causes the data protection information for the statement to be physically written to the Data Protection Log at the time the statement is processed.



Note: Use of the STATUS option is not recommended. See section *Status Protection Option* in chapter *Concepts and Facilities* of the *Adabas Command Reference Manual* for more information.

SUFFIX Option

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. Any value here will override values specified with the global parameter **OPTIONS** or taken from Predict.

Field name *suffix* in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this option](#) for more information.

The WHENEVER Statement

```
EXEC ADABAS
  WHENEVER { NOT FOUND
            SQLERROR } { CONTINUE
                       GO TO label
                       GOTO label }
END-EXEC
```

The WHENEVER statement is used to control the error handling of the program. It affects the code generated by the Adabas Native SQL preprocessor for handling exception conditions.

The 'WHENEVER NOT FOUND GOTO *label*' statement specifies a label to which the program should jump if the 'no records found' condition occurs as a result of the execution of a COMPARE, FIND, FIND COUPLED or SORT statement.

The 'WHENEVER SQLERROR GOTO *label*' statement specifies a label to which the program should jump if an error response code (response code neither = 0 nor = 3) occurs as a result of the execution of an Adabas Native SQL statement.

The 'WHENEVER ... CONTINUE' statement causes the Adabas Native SQL preprocessor to stop generating test-&-branch code after each ADABAS Native SQL statement.

If a 'WHENEVER SQLERROR ...' statement is coded, it deactivates the error handling routine of the standard abort module. You should normally use the SQLERROR together with ABORT.

The variables ISN, QUANTITY and RESPONSE_CODE (Ada, COBOL and PL/I unless the global parameter 'ABORT .' is coded) or SQLISN, SQLQTY and SQLRSP (Ada, COBOL and PL/I if the global parameter 'ABORT .' is coded; also FORTRAN) contain the values from the most recent ADABAS Native SQL statement. These can be used for error analysis.

See sections [Additional Fields in the Record Buffers](#) and [Response Code Interpretation](#) for more information.

See also description of the [ABORT](#) parameter for more information on error processing.

The WRITE TO LOG Statement

```
EXEC ADABAS
  WRITE TO LOG
    USERDATA= var
END-EXEC
```

This statement is used to write user data to the Adabas data protection log. This data may be read and displayed with the ADASEL utility program. See the *Adabas Utilities Manual* for more information.

USERDATA Clause

```
USERDATA= var
```

The data to be written must be stored in the variable denoted by *var*. The variable name must be immediately preceded by a colon (':'), for example 'USERDATA = :NAME'. The length of the user data, that is, the number of characters to be written, must not exceed the limit specified in the USERDATA clause of the global parameter. **OPTIONS**.

6 USING ADABAS NATIVE SQL STATEMENTS IN TP

PROGRAMS

- COM-PLETE 224
- Customer Information Control System (CICS) 224

This chapter describes the procedures that must be observed when writing teleprocessing application programs under COM-LETE, CICS or UTM that issue Adabas Native SQL statements.

No special precautions need to be taken when writing programs that are to run under BS2000/OSD/RTIO, z/VM, TSO or equivalent compatible systems. Programs should be coded in exactly the same way as batch programs.

See also the *Adabas Programmer's Guide* for Teleprocessing Applications.

This chapter covers the following topics:

COM-LETE

TP application programs that are to run under the control of Software AG's COM-LETE TP monitor should be coded in exactly the same way as batch programs.

The COM-LETE utility program USCHC can be used to set the default hard-copy device to 0, so that output produced by DISPLAY statements will be sent to the user's terminal.

Customer Information Control System (CICS)

The CICS Transaction Work Area (TWA) provides a standardized interface for passing parameters to the program. The first six words of the TWA are used by Adabas Native SQL for communication with CICS. Alternatively, the user may choose to use the COMMAREA. Refer to the global parameter **MONITOR**.

The CICS command level interface for Ada, COBOL, FORTRAN and PL/I ensures that programs written in these languages will be quasi-reentrant.

Programs can be written in CICS pseudo-conversational mode with the aid of the SAVE and RESTORE statements. Programs that use this facility must not use the ISNSIZE option.

Adabas Native SQL provides an easy way of defining parameters for generating the CICS code. For further information, see the global parameter **MONITOR**.

See also the global parameter **CICS STUB**.

Passing Parameters to Adabas

The addresses of the Adabas control block, format buffer, record buffer, search buffer, value buffer and ISN buffer are passed in the same manner for all releases of CICS. These addresses must be placed in the first six words of the TWA. Software AG provides an Assembler subroutine, ADASTWA, which places the parameter address in the TWA. The Adabas/CICS interface routine,

ADALNC, retrieves these addresses from the TWA. This module must be used instead of the standard Adabas interface routine, ADALNK. The Ada, COBOL, FORTRAN or PL/I program should call ADASTWA with the TWA as the first parameter; the next six parameters are the customary parameters as used with Adabas direct calls.

Compiling and Executing Adabas Native SQL/CICS Programs

CICS applications programs that use Adabas Native SQL statements must be processed in the following order:

1. Run the program through the Adabas Native SQL preprocessor;
2. Run the program through the CICS preprocessor;
3. Compile the program in the normal manner;
4. Link-edit the program. An INCLUDE statement must be coded to force the inclusion of the subroutine ADASTWA (Ada, COBOL and PL/I) or ADATWA (FORTRAN);
5. Execute the program.

COBOL TP Programs Using Adabas Native SQL and CICS (Command Level)

The following global option parameters must be specified when preprocessing COBOL programs:

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

The ADACALL parameter causes each 'CALL ADABAS' statement to be replaced by a 'CALL ADASTWA' statement. The TELE parameter causes the CICS command level instruction to be inserted after every Adabas command. The ABORT parameter causes the call to the response code analysis module 'RESPCICS' to be called in a "CICS" way.

Alternatively 'MONITOR CICS.' may be used.

Also, the TWA must be declared in the linkage section of the program and the address of the TWA must be made available.

FORTRAN Programs Using Adabas Native SQL and CICS (Command Level)

The code for FORTRAN programs is identical to that for COBOL; however ADASTWA (supplied in the Adabas source library) must be changed to ADATWA.

PL/I TP Programs Using Adabas Native SQL and CICS (Command Level)

The following global option parameters must be specified when preprocessing PL/I programs:

```
ADACALL ADASTWA USING TWA.  
TELE "EXEC CICS LINK PROGRAM ('ADABAS');".  
ABORT RESPCICS CICS.
```

The ADACALL parameter causes each 'CALL Adabas' statement to be replaced by a 'CALL ADASTWA' statement. The TELE parameter causes the CICS command level instruction to be inserted after every Adabas command. The ABORT parameter causes the call to the response code analysis module 'RESPCICS' to be called in a "CICS" way.

Alternatively, 'MONITOR CICS.' may be used.

Also, the TWA must be declared and its address must be made available.

If you implement a multiple-step transaction under UTM, the contents of the control block are lost. You should therefore use the SAVE and RESTORE statements before and after every screen-IO. Also, Adabas must be running in get-next mode, this means specify no ISNSIZE.

7 GLOBAL PARAMETERS

▪ The ABORT Parameter	229
▪ The ADACALL Parameter	232
▪ The APOS Parameter	234
▪ The CICS STUB Parameter	235
▪ The LANG Parameter	236
▪ The LIBRARY Parameter	237
▪ The MODE Parameter	238
▪ The MONITOR Parameter	240
▪ The NAME Parameter	243
▪ The NETWORK Parameter	244
▪ The OPTIONS Parameter	245
▪ The SYSFILE Parameter	256
▪ The TELE Parameter	257
▪ The USER Parameter	258
▪ The VIRTUAL-MACHINE Parameter	259
▪ The XREF Parameter	260

Adabas Native SQL provides a range of global parameters that can be used to define processing options and adapt them to your particular requirements. The options are specified in a parameter file, which is typically included in the job control stream and read by an '//ADAGLOB DD *' JCL card or equivalent.

This chapter lists the global parameters and describes their syntax and their effect.

Note that each of these parameters is terminated by a period ('.').

Global parameters can now contain comment lines. Comment lines are signified by two asterisks ('**') starting in column 1.

This chapter covers the following topics:

The ABORT Parameter

```

ABORT [ module-name ]
        [ IDENT ]
        [ PLI ]
        [ CICS ]
        [ FILE= file-number ]
        [ DBID= database-number ] .

```

The ABORT parameter is used to modify Adabas Native SQL's action when an Adabas command returns a response code other than 0 or 3.

See also [Response Code Interpretation](#) and the *Adabas Messages and Codes Manual*.

In the absence of an ABORT parameter, the abort module RESPINT (Ada, COBOL or PL/I) or RESPF (FORTRAN) is called. This module interprets the response code and prints the appropriate text from the ABEND error message file, the content of the CONTROL-BLOCK and the line sequence number of the erroneous source statement in the SYSOUT file, calls the appropriate trace module, issues an Adabas BT command, and closes the database. Finally, it ABENDs the run.

In particular, the following fields are passed to the error-handling routine:

```

CONTROL-BLOCK
DDFILE
CSEQ
FORMAT-BUF
RECORD-BUF
SEARCH-BUF
VALUE-BUF
CLN1
CLN2
TRCE
CLNNUM
DDDBID

```

CLN1 and CLN2 are arrays that contain the Adabas Native SQL statement. CLN1 contains characters 1..40 of each statement and CLN2 contains characters 41..80. CLNNUM is a variable that indicates the number of elements used in each of these two arrays.

If you want to trap certain error conditions and handle them differently, you must write your own error handling routine. Adabas Native SQL will generate calls to that module instead of to RESPINT if an ABORT parameter with the appropriate module-name is executed. The fields listed above are passed to the module.

If 'ABORT FILE=0.' is coded, Adabas Native SQL does not generate an OPEN for the Natural system.

If ABORT is coded with no module-name, i.e., 'ABORT .', Adabas Native SQL will not check the response code after executing Adabas commands and no exception handling routine will be called. You must write inline code following each Adabas Native SQL statement to handle exception conditions, or use the WHENEVER statement. In addition, if ABORT is coded with no module-name, Adabas Native SQL generates three global fields with the names SQLISN, SQLQTY and SQLRSP instead of generating the three fields ISN, QUANTITY and RESPONSE_CODE for each record buffer.

See also [Additional Fields in the Record Buffers \(Ada, COBOL, PL/I\)](#). (In FORTRAN programs, since there are no record buffers, Adabas Native SQL always generates the above-mentioned three global fields.)

IDENT Clause

If the ABORT parameter is used with the IDENT keyword, Adabas Native SQL generates a statement of the form:

```
CALL identifier ...
```

where the variable identified by the identifier contains the name of the error-handling routine. Otherwise, a statement of the form:

```
CALL 'module-name' ...
```

is generated, where the name of the error-handling routine appears as a literal constant in the CALL statement.

The first form is used for dynamic calls, the second for static calls.

This option is only available in COBOL programs, and it is not supported by all COBOL compilers.

PLI Clause

If the user-written module is in PL/I, the keyword 'PLI' must be coded.

CICS Clause

This clause specifies that the calling mechanism to the response code analysis routine should be generated for CICS. Hence Adabas Native SQL will generate the following:

```
CALL 'ADASTWA' USING ADASQL-LINK-ADDRESSES CONTROL-BLOCKxxxx
DDFILE CSEQ FORMAT-BUFxxxx RECORD-BUFxxxx SEARCH-BUFxxxx VALUE-BUFxxxx
CLN1 CLN2 TRCE CLNNUM DDBID
CALL 'ADASTWA' USING TWA ADASQL-LINK-ADDRESSES
EXEC CICS LINK PROGRAM ('RESPCICS') END-EXEC
```



Note: The definition of ADASQL-LINK-ADDRESSES is generated by Adabas Native SQL, and the user should define the TWA as a 24-byte string.

FILE Clause

If the error texts reside in a file other than the standard Natural system file (FNAT), the FILE clause should be used to specify the file number. This number will be passed to the response code interpretation routine as the second parameter, DDFILE. If FILE=0 is coded, no OPEN command will be issued.

The error texts are commonly stored in the Natural system file (parameter FNAT in the SYSFILE statement).

DBID Clause

This clause may be used to specify a database where the FNAT exists in another environment. RESPINT now accepts another parameter DDBID which has the database number of the FNAT or zero (if the DBID clause is not specified).

The ADACALL Parameter

```
ADACALL module-name [IDENT] [USING id1] [LAST id2].
```

The ADACALL parameter is used to instruct Adabas Native SQL to generate non-standard Adabas calls. Instead of the standard call:

```
CALL 'ADABAS' USING CONTROL-BLOCK... etc.
```

Adabas Native SQL will generate a call as follows:

```
CALL 'module-name' USING id1... CONTROL-BLOCK... etc.
```

The subroutine name 'ADABAS' is replaced by the specified module name in each executable command generated by Adabas Native SQL.

This parameter is used mainly in teleprocessing (TP) applications programs, where the user must call the ADASTWA module. The first parameter of the 'CALL' statement is the terminal work area (TWA); this is followed by the Adabas buffers.

A TP program should therefore specify the following ADACALL parameter:

```
ADACALL ADASTWA USING TWA.
```

This will cause Adabas Native SQL to generate the following call instead of 'CALL 'ADABAS'':

```
CALL 'ADASTWA' USING TWA CONTROL-BLOCK... etc.
```

The ADACALL parameter may also be useful in installations that maintain an I/O interface between the application and Adabas. The ADACALL parameter can be used to direct the calls to the I/O interface, instead of to Adabas.

In CICS environments, the ADALNK module must be replaced by the ADALNC module, which fetches the Adabas parameters (control block, record buffer, etc.) from the TWA.

See also the description of the 'MONITOR' and 'TELE' parameters.

CICS users should also refer to [Using Adabas Native SQL Statements in TP Programs](#).

IDENT Clause

If the ADACALL parameter is used with the IDENT keyword, Adabas Native SQL generates a statement of the form:

```
CALL identifier ...
```

where the variable identified by the identifier contains the name of the Adabas link routine. Otherwise, a statement of the form:

```
CALL 'module-name' ...
```

is generated, where the name of the Adabas link routine appears as a literal constant in the CALL statement. The first form can be desirable in certain circumstances. This option is only available in COBOL programs, and it is not supported by all COBOL compilers.

LAST Clause

The LAST clause is used to specify the seventh parameter generated for the Adabas call. id2 is a structure generated by Adabas Native SQL. It can be modified by the user as desired.

The seventh parameter is only an option of Adabas. It contains information that can be evaluated by an Adabas user exit. Adabas Review uses the seventh parameter to receive information on the program name and library name. Adabas Native SQL put the value of the program name within the structure. The user should plug in the library name, using a simple MOVE statement into the L-variable field. The last clause also causes Adabas Native SQL to generate code that may enable Review to identify the use of the seventh parameter.

The LAST clause of the ADACALL parameter generates a structure with the following names (not applicable to VMS or UNIX):

```
01 Variable
   02    FILLER PIC x(276).
   02    PR-variable PIC x(8)VALUE
        'program'.
   02    L-variable PIC x(8) VALUE
        'library'.
   02    RE-variable PIC x(76).
```

The APOS Parameter

APOS NO .

If the APOS parameter is set to 'NO', character strings generated by Adabas Native SQL will be enclosed in double quotes ("). If the APOS parameter is not coded, character strings will be enclosed in single quotes, sometimes known as apostrophes (').

The CICS STUB Parameter

CICS STUB .

This global parameter is used to improve performance of interpartition commands when using CICS.

In this case, the call is made to the module "Adabas", supplying as the first parameter the stub pointer. Adabas Native SQL generates the definition of the stub pointer, and the user should supply the name by using the ADACALL parameter:

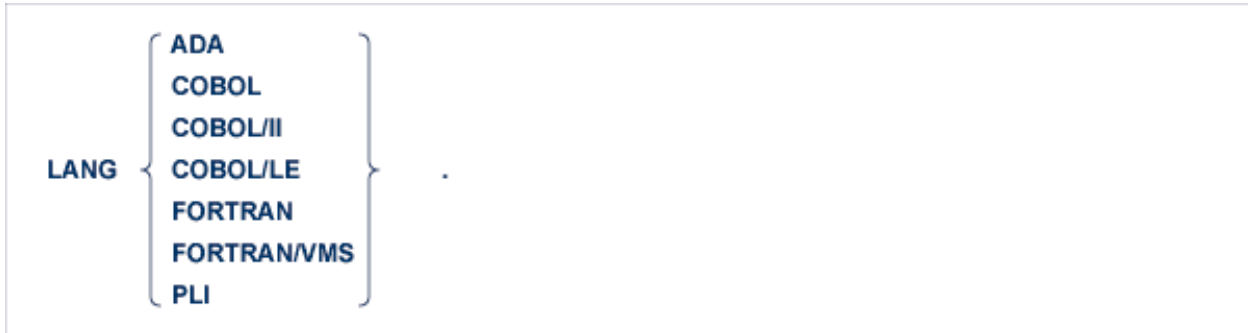
```
ADACALL ADABAS USING pointer
```

With the CICS stub, the user should also use the ABORT CICS parameter if the response code analysis routine should be invoked for CICS.



Note: The user should define the TWA for this purpose.

The LANG Parameter



Adabas Native SQL generates declarations and code in the language specified by this parameter. The code generated with the setting 'COBOL' is also compatible with the COBOL/II compiler, but the code generated with the 'COBOL/II' setting makes use of the structured 'END-IF', 'END-PERFORM', etc., clauses.

If this parameter is omitted, Adabas Native SQL attempts to determine the language in which the program is written by examining its first line. However, this technique is not completely reliable, so we strongly recommend you include this parameter in every Adabas Native SQL run.

The LIBRARY Parameter

```
LIBRARY module-name .
```

This new parameter is used to support a library concept for 3GL applications. *name* represents a logical library name (max. 8 characters). If the library is not defined in Predict, an error message is displayed.

The MODE Parameter

```
MODE [ { FLOW  
        NOFLOW } ] [ NOUPDATE ] [ TRACE ] .
```

This parameter controls debugging facilities that are built in to Adabas Native SQL.

MODE FLOW

If the parameter 'MODE FLOW' is specified, all Adabas Native SQL statements will be printed out at runtime as they are executed.

MODE NOFLOW

If the parameter 'MODE NOFLOW' is specified, the code that copies Adabas Native SQL source statements into a buffer is not generated. This reduces the size of the generated Ada, COBOL, FORTRAN or PL/I code, but the FLOW and TRACE facilities are not available and Adabas Native SQL cannot print out the source statement if a runtime error is detected. This could make debugging more difficult.

MODE NOUPDATE

If MODE NOUPDATE is coded, statements that would modify the database (DELETE, INSERT, UPDATE) have no effect.

MODE TRACE

This parameter must be coded if diagnostic output is required. Conversely, when a program has been debugged and diagnostic output is no longer required, you can delete this parameter and recompile the program. The resulting object module will be smaller and will run faster.

Diagnostic output is controlled by the following:

- the global parameter 'MODE TRACE.'
- the Adabas Native SQL statements 'TRACE ON' and 'TRACE OFF', and
- the value contained in the variable TRCE (Ada, COBOL, PL/I) or SQDE00 (FORTRAN).

The action of the global parameter 'MODE TRACE' is described above.

When processing an Ada, COBOL, FORTRAN or PL/I program, and assuming that the global parameter 'MODE TRACE' has been coded, Adabas Native SQL only generates the code for pro-

ducing diagnostic output when it encounters a 'TRACE ON' statement, it stops generating this code when it encounters a 'TRACE OFF' statement. These two statements provide static control of the diagnostic output, that is, they control the section or sections of the program in which diagnostic code is generated.

When an Adabas Native SQL statement is executed, the first action of the diagnostic code is to test the value contained in the variable TRCE (Ada, COBOL, PL/I) or SQDE00 (FORTRAN). If this value is 'OFF', then no further action is performed. Otherwise, the statement is printed out together with the contents of the buffers. This variable provides dynamic control of the diagnostic output. By assigning values to this variable at runtime, you have greater control over the diagnostic output. For example, you could limit output to the first five executions of a loop that may be executed several hundred times.

The MONITOR Parameter

```
MONITOR CICS [ twa ] [ COMMAREA commarea-name POINTERS addr-name ] .
```

This parameter makes it unnecessary to code the ADACALL and TELE global parameters.

If the optional *twa* clause is coded, this name is used instead of the default name 'TWA'.

For COBOL programs, coding 'MONITOR CICS.' is equivalent to coding the following three global parameters:

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

The MONITOR CICS parameter is not valid in FORTRAN programs.

For PL/I programs, coding 'MONITOR CICS.' is equivalent to coding the following three global parameters:

```
ADACALL ADASTWA USING TWA.  
TELE "EXEC CICS LINK PROGRAM ('ADABAS') ;".  
ABORT RESPCICS CICS
```

These defaults may be overridden by coding one or both of the ADACALL or TELE parameters.

Prior to all calls created by the ADACALL parameter, the following code will be generated for COBOL if 'MONITOR CICS.' has been coded:

```
MOVE xxxxxxxxxxx to ADASQL-SAVE-TWA
```

The corresponding code for PL/I programs is:

```
ADASQL_SAVE_TWA = xxxxxxxxxxx
```

←

xxxxxxxxxx is the field name supplied after USING in the global parameter ADACALL. (TWA is the default ADACALL used.)

After each TELE line is generated, the following COBOL code is inserted if 'MONITOR CICS.' has been coded:

```
MOVE ADASQL-SAVE-TWA TO xxxxxxxxxxx
```

The code for PL/I programs is:

```
xxxxxxxxxxx = ADASQL_SAVE_TWA
```

If you are using more than 28 bytes in the TWA, code the following:

```
01 TWA.
      02 ADABAS-TWA          PIC X(28)
      02 REST-OF-TWA        PIC X(nnnn).
```

Then code the following ADACALL parameter for COBOL:

```
ADACALL ADASTWA USING ADABAS-TWA
```

We recommend defining the layout of the TWA in COBOL copy books which can be accessed by all Adabas Native SQL programs.

The code for PL/I is:

```
DCL 01 TWA,
      02 ADABAS_TWA          CHAR(28)
      02 REST_OF_TWA        CHAR(nnnnn);
```

This parameter also controls the generation of EXEC CICS LINK to RESPCICS and PRTRCICS instead of RESPINT and PRTRACE.

The COMMAREA parameter is for using the COMMAREA instead of the TWA.

The user then will have to define a structure for the Commarea usage as follows:

```
01 COMMAREA-NAME.
02 FILLER PIC X(8) VALUE 'ADABAS52'.
02 ADDR-NAME.
03 FILLER PIC X(4) OCCURS 6.
```

Adabas Native SQL will then generate

- a call to ADASTWA with addr-name to move the addresses of the Adabas buffers into it;

- and then an EXEC CICS command with COMMAREA(*commarea-name*) instead of TWA.

The constant "ADABAS52" is the indicator for the Adabas CICS interface to detect the COMMAREA parameter list. For the syntax of the Adabas parameter list, see also the Adabas CICS command level interface description for CICS Version 3.2 and higher.



Note: The RESPCICS and PRTRCICS will continue to use the TWA.

The NAME Parameter

```
NAME program-name .
```

The program-name specified in the 'NAME' parameter is used by Adabas Native SQL in conjunction with the programming language (Ada, COBOL, FORTRAN or PL/I) when Adabas Native SQL writes Xref data to the data dictionary. The *program-name* is referred to in Predict as *Member*.

Adabas Native SQL provides cross-reference reports of programs, modules and fields using the Xref facilities of Predict. This information is automatically created during the preprocessor pass. The names of the files and fields that are used are taken from the FROM, SELECT, WHERE, SET, etc., clauses of the Adabas Native SQL statements; the name of the program that uses them is taken from the 'NAME' parameter.

If the NAME parameter is omitted, Adabas Native SQL takes the program name from the following sources:

Language	Program name taken from
ADA	the procedure
COBOL	PROGRAM-ID paragraph in the environment division
FORTRAN	the first line of the program, which must be PROGRAM <i>programe</i>
PL/I	the label preceding 'PROC OPTIONS(MAIN)'

The NETWORK Parameter

```
NETWORK network-name .
```

This global parameter defines the network in which the program is to run. *network-name* must be defined in Predict, and must be linked to the virtual machine specified with the parameter **VIRTUAL-MACHINE**.

This parameter is mandatory, if one or more networks other than HOME are defined in Predict.

A network contains all virtual machines and databases that are to be accessed. In fact, all databases that are used in the program should belong to the network specified here.

For every database used (DBID, AUTODBDID, AUTODBDID-ATM or AUTODBDID-ALL clauses), Adabas Native SQL checks that if the database is defined as local it belongs to the current virtual machine, and if the database is isolated that it belongs to the current network.

The OPTIONS Parameter

OPTIONS

- [ADA-VERSION= { 62 | 71 }]
- [{ AUTODBID
AUTODBID-ATM
AUTODBID-ALL
DBID= *database-name* }]
- [BINARY]
- [COND-NAME= { Y | N }]
- [DYNAMCID]
- [GFORMAT]
- [INDEXED= { Y | N }]
- [INIT-LOW-VALUE]
- [ISNSIZE= *len1*]
- [LONG-COUNTER]
- [LARGE-NUMBERS]
- [NEW-CONTROL-BLOCK]
- [NONDE= { Y
N
D }]
- [OLDCOND-NAME]
- [OPEN]
- [PREFIX= *prefix*]
- [SOFT= { Y | N }]
- [STATIC= { Y | N }]
- [SUFFIX= *suffix*]
- [TRUNCATION= { L
M
R }]
- [USERDATA= *len2*]
- [VALIDATION= { *
' *character* '
" }] .
- [VISTA]

The `OPTIONS` parameter enables the user to specify various processing options that will take effect for the whole of the program unless they are overridden by declarations made at the individual statement level.

The `OPTIONS` parameter should not be confused with the `OPTIONS` clause of individual Adabas Native SQL statements.

ADA-VERSION Clause

The `ADA-VERSION` clause indicates to Adabas Native SQL in which Adabas version the precompiled program is to be executed. The default is 62, and this will generate code that can be executed in all Adabas versions. The value 71 will enable using new features introduced in Adabas Version 7.1 in the `READ LOGICAL` and `HISTOGRAM` statements.



Note: A precompiled program with `ADA-VERSION=71` may fail or give unpredicted results if executed in an Adabas version lower than 7.1.

AUTOBID Clause

The `AUTOBID` option causes every access statement to use the database identified in Predict for that file. If the file is linked to a database and no specific `DBID` is specified in the statement, an error message is given.

AUTOBID-ALL Clause

The `AUTOBID-ALL` option causes all statements (both access and update) to use the database identified in Predict for that file. If the file is linked to a database and no specific `DBID` is specified in the statement, an error message is issued.

If `AUTOBID-ALL` is specified, neither the `DBID` nor the `AUTOBID` clause may be used.

The following rules apply to the various statements when using the `AUTOBID-ALL` clause:

Statement	Remarks
All statements	<p>One file must be documented in exactly one database.</p> <p>If separate test and production environments are used, separate dictionary files (FDIC) are necessary.</p> <p>No source changes are necessary if only a recompile with the production dictionary is required.</p>
CONNECT, DBCLOSE	<p>These statements are mandatory and must be used within one program.</p> <p>The files specified in the <code>UPDATE</code> clause define the database to be updated. All update files must be within one database otherwise an error message is displayed. If more</p>

Statement	Remarks
	<p>than one database is accessed, several OPEN commands must be generated, but only one OPEN command is generated for update.</p> <p>To generate a CLOSE command to the same databases which are opened using CONNECT, the DBCLOSE must occur within the same program, otherwise an error message is given.</p>
COMMIT	The COMMIT statement is always sent to exactly one database. This database is identified by the CONNECT statement (updatefiles => database) or by the UPDATE statements available in the program. An UPDATE or CONNECT statement must be coded, otherwise an error message is given.
UPDATE, DELETE, INSERT	<p>If the program does not contain a CONNECT statement, a warning is issued that CONNECT must be executed before the updates are performed, otherwise consistency cannot be guaranteed. Adabas Native SQL must be able to check that within one program updates are performed only on files that belong to the same database.</p> <p>The update of only one database is supported.</p>

AUTO DBID-ATM Clause

The AUTO DBID-ATM option causes all statements (both access and update) to use the database identified in Predict for that file. If the file is linked to more than one database and no specific DBID is specified in the statement, an error message is issued.

If AUTO DBID-ATM is specified, neither the AUTO DBID-ALL nor the AUTO DBID clause may be used.

With this option we do not restrict the number of updated databases (unlike the AUTO DBID-ALL parameter) and the user does not have to specify any DBID in any of the statements.



Note: Please note that if such a program will be run without the supervision of the Adabas Transaction Manager, we cannot guaranty the data integrity and this will be the user’s responsibility to ensure the use of the Adabas Transaction Manager.

The Commit statement will be generated with the default DBID and the Adabas Transaction Manager will take care of the two phase commit.

If within this program the user would like to use the CONNECT statement, then he should specify in this statement the DBID to which this CONNECT should run. The same would apply to the DBCLOSE statement.

BINARY Clause

This clause applies to COBOL programs only.

It will cause all binary fields to be generated as BINARY instead of COMP.

COND-NAME Clause

This clause applies to COBOL programs only.

If the clause 'COND-NAME = Y' is coded, the record buffer generated by Adabas Native SQL includes the condition names defined in Predict as level-88 entries.

This global value will be overridden by any value specified in a clause of an individual Adabas Native SQL statement.

The field `With Cond. names` in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See also *Generate COBOL Copy Code* in the *Predict Administration Manual*.

DBID Clause

This clause should be used if the program accesses more than one database. The database-name must be defined in the data dictionary, and the data dictionary description of the database must include the file or files to be accessed. All statements including UPDATE, DELETE and STORE are affected by this clause.

DYNAMCID Clause

If the DYNAMCID keyword is coded, Adabas Native SQL generates the command IDs of the Adabas control blocks dynamically during program execution.

The automatic Adabas routine for generating the command ID is used. Using DYNAMCID increases performance significantly.

If this clause is not specified, the command ID used for each Adabas command is generated from the cursor-name of the corresponding Adabas Native SQL statement. If the DYNAMCID keyword is not coded in the global OPTIONS parameter and a cursor-name is not defined for a particular Adabas Native SQL statement, because the DECLARE option was not used, Adabas Native SQL will generate command IDs in the form -m-n, where mn is a sequence number starting from 01. The first statement without a DECLARE clause will have command ID -0-1, the second statement will have -0-2, etc.

The command ID is used by Adabas for the following purposes:

- As an identifier for the internal, decoded version of the format buffer. Efficiency is improved if Adabas statements that use the same format buffer use the same command ID, otherwise Adabas is compelled to re-interpret the format buffer each time.
- When executing HISTOGRAM, READ LOGICAL and READ PHYSICAL SEQUENCE statements. If the command ID is not given when the statement is executed, Adabas 'loses its place' in the file and gives inconsistent results.
- To identify ISN lists. The command ID links the Adabas command (COMPARE, FIND, FIND COUPLED, or SORT) that creates the ISN list with subsequent commands that retrieve the records whose ISNs are stored in the list.

If several programs that use Adabas Native SQL statements are linked together, all command IDs must be unique. This can be achieved explicitly, that is, by coding a unique cursor-name for each statement, or by allowing Adabas Native SQL to allocate the command IDs dynamically by means of the DYNAMCID global option. Coding unique cursor-names has the advantage that the Adabas command log is easier to interpret.

See section *Command ID Usage* in the *Adabas Command Reference Manual* for more information.

GFORMAT Clause

This clause indicates that a global format is to be generated for this program. Adabas Native SQL generates a unique global format ID for every declaration generated (with the exception of variable index used for periodic groups or multiple-value fields). The global format ID is unique and will not exist in other programs. This clause can help to improve application performance, particularly in on-line environments, by reducing the number of format buffer translations that Adabas has to perform.

If this option is used, the global format ID is generated from the following information:

GFID = *abcdeef*

Where for FDIC file number and DBID < 255,

<i>a</i> = x'C0'
<i>b</i> = x'83'
<i>c</i> = FDIC DBID
<i>d</i> = FDIC FNR
<i>eee</i> = Adabas Native SQL sequence number from Predict defaults
<i>f</i> = An internal sequence number within the program

Where for FDIC file number or DBID > 255

a = x'C1'
b = Possible value x'00' to x'FF'
c = Right byte of FDIC DBID
d = Right byte of FDIC FNR
eee = Adabas Native SQL sequence number from Predict defaults
f = An internal sequence number within the program

The GFORMAT clause is not available in Ada programs.

INDEXED Clause

This clause applies to COBOL programs only.

If the INDEXED clause is specified, all multiple-value fields and periodic groups are generated with the 'INDEXED BY' keywords. The name of the index is taken from Predict. If no index name is defined in the data dictionary, the name of the multiple-value field or periodic group is used, prefixed with 'I'.

This global value will be overridden by any value specified in a clause of an individual Adabas Native SQL statement.

Indexed by in the Predict Modify COBOL Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this clause](#) and section *Generate COBOL Copy Code* in the *Predict Administration Manual* for more information.

INIT-LOW-VALUE Clause

By default, Adabas Native SQL is generating the Value buffer fields with an initial value of blanks for alphanumeric fields and zeroes for numeric fields. In this way a Read logical command without a WHERE clause will start the sequential read from those starting values.

With this option the generated alphanumeric fields in the Value buffer will have an initial value of X'00' which has a collating sequence lower than blanks. In this way if the descriptor which we use for the Read logical has values lower than blanks and we don't specify the WHERE clause we will start the sequential read from the lowest value available for this field.

Please note that this feature is available for Cobol and PL/1 languages only.

ISNSIZE Clause

If the ISNSIZE clause is specified, the default size of the ISN buffer is defined and all Adabas Native SQL statements run in 'ISN buffer' mode; however, the buffer size can be modified for individual retrieval statements by local ISNSIZE specifications.

If a global ISNSIZE value is not specified, ISN buffers are allocated for individual statements as determined by the presence or absence of the ISNSIZE parameter in the OPTIONS clause of each individual statement.

ISN buffer mode must not be used when accessing files that use the 'security by value' facility.

See also *ISN Lists and the ISN Buffer*.

LARGE-NUMBERS Clause

This option will cause Adabas Native SQL to generate numeric (Unpacked) and Packed fields in Cobol for numeric fields with up to 31 digits. Without this option numeric fields with more than 18 digits will be generated as a character string. Users should note that in case of using the LARGE-NUMBER clause, they should make sure that the Cobol compiler option that allows for up to 31 digits numeric fields is set.

LONG-COUNTER Clause

This option will cause Adabas native SQL to generate the Multiple Value and Periodic Group counters as 4 binary bytes instead of the default of 2 binary bytes. This option should be used if the total occurrences of the Multiple Value or the Periodic group may exceed 32767 occurrences.

NEW-CONTROL-BLOCK Clause

This option will cause Adabas Native SQL to generate the new control block structure introduced in Adabas 6.1 on mainframe platforms and Adabas 4.1 in OpenVMS.

The new control block will allow file numbers and dbid's to be greater than 255.

NONDE Clause

This clause is available with Adabas 5 only. It is used to allow (NONDE=Y) or inhibit (NONDE=N) the use of non-descriptors within database search criteria. The option NONDE=D specifies that each search criterion must include at least one descriptor (and possibly some non-descriptors).

The field `Non-descriptor search allowed` in the Predict Modify Adabas Native SQL Defaults screen must be set to "Y" if you want to use this option.

OLDCOND-NAME Clause

Adabas Native SQL allows for the condition values to contain blanks. In versions prior to V231, a blank in the value of the condition name definition in Predict was considered as a delimiter and Adabas Native SQL generated several values for the same condition name value line.

With version 231 and up, only one value will be generated per value line definition in Predict and this value may contain blanks.

Users that would like to keep the old functionality may use this keyword.

With this option, Adabas Native SQL will generate the Condition names as in versions prior to V231 and consider a blank in the value to be a delimiter.

OPEN Clause

If this clause is coded, Adabas Native SQL performs an explicit 'open' on Predict file as it preprocesses your application program.

PREFIX Clause

If the option 'PREFIX = *prefix*' is coded, the field names generated for the record buffer will include the specified prefix. This global value will be overridden by any value specified in a clause of an individual Adabas Native SQL statement or taken from the data dictionary.

Field name `prefix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this clause](#) for more information.

SOFT Clause

This clause is used to enable (SOFT=Y) or inhibit (SOFT=N) the soft-coupling option.

The field `Use of soft-coupling allowed` in the Predict Modify Adabas Native SQL Defaults screen must be set to "Y" if you want to specify this option.

STATIC Clause

This option applies to PL/I programs only.

If the option 'STATIC = Y' is coded, all buffers generated by Adabas Native SQL will be defined as static. This global value will be overridden by any value specified in a clause of an individual Adabas Native SQL statement.

The field `Static` in the Predict Modify PL/I Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this clause](#) for more information.

SUFFIX Clause

If the option 'SUFFIX = *suffix*' is coded, the field names generated for the record buffer will include the specified suffix. This global value will be overridden by any value specified in a clause of an individual Adabas Native SQL statement or taken from the data dictionary.

Field name `suffix` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option. See the [previous discussion on this clause](#) for more information.

TRUNCATION Clause

A field name may exceed the maximum number of characters permitted by the language, particularly if a prefix and/or suffix has been added. Adabas Native SQL uses the TRUNCATION clause to delete excess characters:

TRUNCATION = L	truncate from the left
TRUNCATION = M	truncate from the middle
TRUNCATION = R	truncate from the right

This global value will override the value in the data dictionary.

The field `Truncation` in the Predict Modify...Defaults screen must be marked with an "X" if you want to specify this option.

USERDATA Clause

The USERDATA clause may be used to specify the size of the ET-data buffer, i.e., RECORD-BUFOPN. The default size is 500 bytes. This buffer is used in COMMIT WORK, DBCLOSE and CHECKPOINT statements.

VALIDATION Clause

This option determines how invalid characters in field names - including prefix and suffix, if specified - are handled by Adabas Native SQL.

Validation Character	Result
Null string (two consecutive apostrophes)	Invalid characters in a field name will result in an error message but will not be modified.
Replace character (letters A-Z, digits 0-9 or special character depending on language)	Invalid characters in a field name are replaced by this character.
Asterisk	Invalid characters in the field name are deleted.

This global value will override the value in the data dictionary.

Validation in the Predict Modify...Defaults screen must be marked with an "X" if you want to use this option.

VISTA Clause

In case that the Predict file is defined under Adabas Vista configuration, the DBID and file number of the physical Predict file may differ from the Adabas Native SQL `SYSFILE` parameter. In this case Adabas Native SQL will issue an error that these numbers do not match the Predict Control record.

The VISTA clause will cause Adabas Native SQL to ignore the different values and use the numbers from the Predict Control record for generating the global format id.

The SYSFILE Parameter

```
SYSFILE  [ FNAT=( dbid1,fnr1 [ ,password1 [ ,cipher1 ] ] ) ]  
         [ FDIC= ( dbid2,fnr2 [ ,password2 [ ,cipher2 ] ] ) ]
```

The SYSFILE parameter specifies to Adabas Native SQL the number of the Natural system file and the number of Predict file. The Adabas Native SQL error messages are normally stored in the Natural system file. The SYSFILE parameter is mandatory, and the *dbid* and *fnr* must be specified. These numbers are checked against the DDA default record in Predict and, in case of incompatibility, execution stops.

PASSWORD Clause

If the Predict file or Natural system file is password-protected, the correct password must be specified using this clause.

CIPHER Clause

If the Predict file or Natural system file is enciphered, the correct cipher key (cipher code) must be specified using this clause.

Example:

```
SYSFILE  FNAT = (3,9)  FDIC = (3,8)
```

The database ID (DBID) of the Natural system file is 3, and its filenumber (FNR) is 9. The DBID of the Predict file is 3, and its FNR is 8.

The TELE Parameter

```
TELE "text" .
```

The TELE parameter specifies a source statement to be inserted after each CALL command in the generated executable statements. The text may, for example, be a command required by a teleprocessing monitor.

Example:

```
TELE "EXEC CICS LINK PROGRAM ('ADABAS') END-EXEC".      (COBOL)
TELE "EXEC CICS LINK PROGRAM ('ADABAS');".              (PL/I)
TELE "EXEC CICS LINK PROGRAM ('ADABAS') END-EXEC".      (FORTRAN)
```

The above example inserts the CICS command level instruction after every CALL. This parameter should be used in conjunction with the ADACALL parameter. See the example in the ADACALL parameter.

There may be up to five TELE statements, so that up to five additional lines of text may be generated after the call.

Note that in COBOL programs the *text* must not include a period.

See also the [MONITOR](#) parameter.

Ada is still not supported by the CICS translator.

The USER Parameter

```
USER userid .
```

This parameter is used to identify the user responsible for the program. This userid will be documented in XREF. If no USER parameter is specified, Adabas Native SQL takes the first 3 characters of the program as the userid.

The VIRTUAL-MACHINE Parameter

```
VIRTUAL-MACHINE virtual-machine .
```

This statement defines the virtual machine, that is the real computer or node, in which the program is to run.

virtual-machine must be defined in Predict, and must be linked as a child object to the network specified with the **NETWORK** parameter.

This parameter is mandatory if one or more databasesVirtual Machines other than HOME are defined in Predict.

For every database used (DBID, AUTODBID, AUTODBID-ATM or AUTODBID-All clauses), Adabas Native SQL checks that if the database is defined as *local* it belongs to the current virtual machine, and if the database is *isolated* that it belongs to the current network.

The XREF Parameter



XREF { ON
OFF
FORCE } .

This parameter controls the writing of cross-reference information (Xref data) to the data dictionary.

Note that the XREF global Adabas Native SQL parameter interacts with Predict's Preprocessor force option. If the Predict option is set to 'Y', then Adabas Native SQL ignores the XREF parameter in the Adabas Native SQL global parameter file (if present) and proceeds as though 'XREF FORCE' had been coded.

For details of the Predict Preprocessor Force option, see section *Common Parameters* in chapter *Generation of the Predict Administration Manual*.

XREF ON.	Adabas Native SQL makes entries in the data dictionary indicating which files and fields are used by the current program.
XREF OFF.	No entries are made in the data dictionary.
XREF FORCE.	Adabas Native SQL makes entries as for 'XREF ON' and additionally checks that the data dictionary contains a program description bearing the name of the current program. An error message is output if this condition is not satisfied.


If the data dictionary is opened for access only (global parameter DDFILE ACC.), 'XREF OFF.' must be coded.

The program is identified by its name (referred to in Predict as *Member*) together with the language in which it is written (Ada, COBOL, FORTRAN or PL/I). See the [NAME](#) parameter.

8

APPENDIX A - SIZE LIMITATIONS

The standard version of Adabas Native SQL is limited to the following maximum sizes:

- Length of an Adabas Native SQL source statement: 100 lines.
 -  **Note:** If the length of the statement exceeds 15 lines, then only the first 11 lines, a line of dots, and the last 3 lines will be stored for the purposes of TRACE, FLOW and runtime error reporting.
- Number of fields mentioned by name in the SELECT clause: 300.
- Number of fields contained in a file that is referenced with SELECT *: 500. (This number includes also the Redefinition fields within this file).
- Number of distinct field names used within the program (these are the field names that will be written into the data dictionary by the cross-reference facility): 2000.
- Number of distinct external subroutines used within the programs (their names will be written into the data dictionary by the cross-reference facility): 500.
- Number of variable indices used within the program: 99.
- Number of elements of redefinitions in one SELECT (PL/I only): 99.
- Number of Adabas Native SQL SELECT statements without a 'CURSOR' : 100
- Number of Adabas Native SQL statements that use 'CURSOR FOR' (multiple record processing): 120 (only applicable to operating systems z/OS, BS2000/OSD and VMS).
- Number of Adabas Native SQL statements that use 'CURSOR' (including statements that use 'CURSOR FOR'): 150 (only applicable to operating systems z/OS, BS2000/OSD and VMS).
- Number of files mentioned in the CONNECT statement: 100.
- Number of declarations without a 'CURSOR': 100
- Number of lines of ADA, COBOL, FORTRAN or PL/I code generated for one Adabas Native SQL statement, not including the direct calls: 200.

- Maximum numbers supported for DBID: 32767.
- Size of format buffer generated: 32767 bytes.
- Number of selection criteria in a 'WHERE' clause: 30.
- Number of constants (literals) used in selection criteria throughout the program: 250.
- Number of constants (literals) used in selection criteria within one Adabas Native SQL statement: 69.
- Number of variables in SET clause of UPDATE/INSERT statement: 300.
- Number of characters in a literal within an Adabas Native SQL statement: 38.

Restrictions in ADA:

- No redefinition.
- No groups generated.
- Periodic groups are always generated with STRUCT=N.
- PACKED and UNPACKED fields are generated as alpha.
- Superdescriptors are not divided into parts in the value buffer.
- The DBID option is not supported.
- FIND COUPLED is not supported.
- The GLOBAL FORMAT-ID option is not supported.

9 APPENDIX B - DESCRIPTIONS OF THE FILES USED IN THE EXAMPLES

■ FORTRAN Synonyms	265
--------------------------	-----

APPENDIX B - DESCRIPTIONS OF THE FILES USED IN THE EXAMPLES

These file descriptions, which are used in the Ada, COBOL, FORTRAN and PL/I examples shown in the following appendices, are supplied on the Predict installation tape. They can be loaded into the data dictionary using the Load function of the migration utility as described in the *Predict Administration Manual*. FORTRAN synonyms that must be used in order for the examples to run are listed at the end of this appendix.

```

> + Fi: EMPLOYEES L: 1 S:
36
Ty L Field name F Length Occ D U DB S
* - - - - - * - - - - - * * - - *
GR 1 PERSONNEL-ID A 8.0 D U AA
GR 1 FULL-NAME AB
2 FIRST-NAME A 20.0 AC N
2 MIDDLE-I A 1.0 AD N
2 NAME A 20.0 D AE
1 MIDDLE-NAME A 20.0 AD N
1 MAR-STAT A 1.0 AF F
1 SEX A 1.0 AG F
1 BIRTH U 6.0 D AH
GR 1 FULL-ADDRESS A1
MU 2 ADDRESS-LINE A 20.0 8 AI N
2 CITY A 20.0 D AJ N
2 ZIP A 10.0 AK N
2 POST-CODE A 10.0 AK N
2 COUNTRY A 3.0 AL N
GR 1 TELEPHONE A2
2 AREA-CODE A 6.0 AN N
2 PHONE A 15.0 AM N
1 DEPT A 6.0 D AO
1 JOB-TITLE A 25.0 D AP N
PE 1 INCOME 40 AQ
2 CURR-CODE A 3.0 AR N
2 SALARY P 9.0 AS N
MU 2 BONUS P 9.0 12 AT N
GR 1 LEAVE-DATA A3
2 LEAVE-DUE U 2.0 AU
2 LEAVE-TAKEN U 2.0 AV N
PE 1 LEAVE-BOOKED 20 AW
2 LEAVE-START U 6.0 AX N
2 LEAVE-END U 6.0 AY N
MU 1 LANG A 3.0 15 D AZ N
PH 1 PHONETIC-NAME A 20.0 D PH
SP 1 LEAVE-LEFT B 4.0 D H1 N
SB 1 DEPARTMENT A 4.0 D S1
SP 1 DEPT-PERSON A 26.0 D S2
SP 1 CURRENCY-SALARY A 12.0 D S3 N
- - - - -

```



```

>
> + Fi: VEHICLES
L: 1 S: 16
Ty L Field name F Length Occ D U DB S All
* - - - - - * - - - - - * * - - *
1 REG-NUM A 15.0 D U AA N
1 CHASSIS-NUM B 4.0 AB F
1 PERSONNEL-ID A 8.0 D AC
GR 1 CAR-DETAILS CD
2 MAKE A 20.0 D AD N
2 MODEL A 20.0 AE N
2 COLOR A 10.0 D AF N
2 COLOUR A 10.0 D AF N
1 YEAR U 2.0 AG N
1 CLASS A 1.0 D AH F
1 LEASE-PUR A 1.0 AI F
1 DATE-ACQ U 6.0 AJ N
1 CURR-CODE A 3.0 AL N
MU 1 MAINT-COST P 7.0 60 AM N
SP 1 DAT-ACQ-DESC B 4.0 D AN
SP 1 MODEL-YEAR-MAKE A 22.0 D AO
    
```

This chapter covers the following topics:

FORTRAN Synonyms

File EMPLOYEES:

PERSONNEL-ID	PID
FIRST-NAME	FNAME
INCOME	INC

File VEHICLES:

PERSONNEL-ID	PID
MODEL-YEAR-MAKE	MOYEMA
REG-NUM	REGNUM

In order to run FORTRAN example 3, the field SALARY must be changed from P9 to I4. The small difference in the total is attributable to rounding in the integer-to-real and real-to-integer conversions.

10 APPENDIX C - ADABAS NATIVE SQL STATEMENTS USED IN THE EXAMPLES

The table below shows which statements are used in each example. For example, the BEGIN and CLOSE statements are used in every example; the COMMIT WORK statement is used in Examples 2 and 3.

The correspondingly numbered Ada, COBOL, FORTRAN and PL/I examples are equivalent.

Example	1	2	3
BEGIN	x	x	x
CLOSE	x	x	x
COMMIT WORK		x	x
CONNECT			x
DBCLOSE	x	x	x
DELETE		x	
FETCH	x	x	x
FIND	x	x	
HISTOGRAM			x
OPEN	x	x	x
READ LOGICAL		x	x
UPDATE		x	x

11 APPENDIX D - ADA EXAMPLES

▪ Example 1	270
▪ Example 2	271
▪ Example 3	274

This chapter covers the following topics:

Example 1

```

with TYPES, ADABAS_GENERIC_CALLS, TEXT_IO ;
use  TYPES, TEXT_IO ;
--
-- AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH
-- CONTAINS FIELDS TAKEN FROM TWO FILES.  THE FIELDS PERSONNEL-ID
-- NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,
-- PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE
-- FOLLOWING CONDITION:
--     PERSONNEL-ID BETWEEN 10000001 AND 19999999
--     MODEL-VEAR-MAKE >
--     CLASS = 'C'
procedure AEX1 is
  START_MODEL : STRING (1..20) := "MERCEDES-BENZ      ";
  START_YEAR_MAKE : STRING (1..2) := "86" ;
  START_MODEL_YEAR_MAKE : STRING(1..22) := START_MODEL &
      START_YEAR_MAKE ;

  FILLE1 : STRING(1..20) := " PERSONNEL-ID          " ;
  FILLE2 : STRING(1..17) := " NAME                  " ;
  FILLE3 : STRING(1..18) := " FIRST-NAME           " ;
  FILLE4 : STRING(1..6) := "BIRTH      " ;
  FILLE5 : STRING(1..3) := "SEX" ;
  HEADER : STRING(1..64) := FILLE1 & FILLE2 & FILLE3 & FILLE4
      & FILLE5 ;
  HEADER2:  STRING(1..64) := (1..64 => '*');
  SPACE_LINE : STRING(1..80) := (1..80 => ' ');
  EXEC ADABAS
  BEGIN DECLARE SECTION
  END-EXEC

  EXEC ADABAS
  DECLARE EMPL CURSOR FOR
  SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX
  FROM EMPLOYEES, VEHICLES
  WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
      AND PERSONNEL-ID BETWEEN "10000001" AND "19999999"
      AND VEHICLES.MODEL-YEAR-MAKE > :START_MODEL_YEAR_MAKE
      AND VEHICLES.CLASS = "C"
  END-EXEC

```

```

begin
    EXEC ADABAS
    OPEN EMPL
    END-EXEC
    PUT_LINE (HEADER) ;
    PUT_LINE (HEADER2) ;
    PUT_LINE (SPACE_LINE) ;

    EXEC ADABAS
    FETCH EMPL
    END-EXEC

    while ADACODE /= 3 loop
        PUT_LINE (" " & EMPLOYEES.PERSONNEL_ID & " " & EMPLOYEES.NAME &
            " " & EMPLOYEES.FIRST_NAME & " " & EMPLOYEES.BIRTH & " "
            & EMPLOYEES.SEX ) ;

        EXEC ADABAS
        FETCH EMPL
        END-EXEC

    end loop ;

    EXEC ADABAS
    CLOSE EMPL
    END-EXEC

    EXEC ADABAS
    DBCLOSE
    END-EXEC
end AEX1 ;

```

Example 2

```

with TYPES, ADABAS_GENERIC_CALLS, TEXT_IO ;
use TYPES, TEXT_IO ;
--
-- DELETE AN EMPLOYEE RECORD AND RELEASE ALL CARS WHICH ARE
-- ASSIGNED TO THIS EMPLOYEE. APRIVATE CAR WILL BE DELETED
-- AND A COMPANY CAR WILL BE MADE A POOL-CAR WHICH IS IDENTIFIED
-- BY ITS PERSONNEL-ID CONTAINING ONLY THE COUNTRY CODE.

procedure AEX2 is
    PERSONNEL_NUMBER : STRING(1..8) := "20007100" ;
    EMPLOYEE_ISN : INTEGER := 0 ;

```

```
EXEC ADABAS
BEGIN DECLARE SECTION
END-EXEC

EXEC ADABAS
READ LOGICAL
DECLARE VEH1 CURSOR FOR
SELECT REG-NUM, PERSONNEL-ID, CLASS
FROM VEHICLES
WHERE PERSONNEL-ID GE :PERSONNEL-NUMBER
OPTIONS HOLD
ORDER BY PERSONNEL-ID
END-EXEC
```

```
begin
--
-- FIND EMPLOYEE
--
EXEC ADABAS
FIND
SELECT
FROM EMPLOYEES EMPLOYEES_1
WHERE PERSONNEL-ID = :PERSONNEL_NUMBER
OPTIONS HOLD
END-EXEC
--
-- IF THE PERSONNEL-ID EXISTS DELETE THE EMPLOYEE AND READ THE
-- VEHICLES FILE

if EMPLOYEES_1.QUANTITY = 1 then

EMPLOYEE_ISN := EMPLOYEES_1.ISN ;
--
-- DELETE EMPLOYEE
--
EXEC ADABAS
DELETE
FROM EMPLOYEES
WHERE ISN = :EMPLOYEE_ISN
END-EXEC
--
-- READ VEHICLES-FILE
--

EXEC ADABAS
OPEN VEH1
END-EXEC

EXEC ADABAS
FETCH VEH1
END-EXEC
```



```
while ADACODE /= 3 AND
  VEHICLES.PERSONNEL_ID = PERSONNEL_NUMBER loop
  if VEHICLES.CLASS = "P" then
    EXEC ADABAS
    DELETE
    FROM VEHICLES
    WHERE CURRENT OF VEHI
    END-EXEC
    PUT_LINE ("PRIVATE CAR" & VEHICLES.REG_NUM &
      "HAS BEEN DELETED");
  else
    VEHICLES.PERSONNEL_ID := VEHICLES.PERSONNEL_ID (1..1)
      & " " & " ";
    EXEC ADABAS
    UPDATE VEHICLES
    WHERE CURRENT OF VEHI
    END-EXEC
    PUT_LINE ( "COMPANY CAR " & VEHICLES.REG_NUM &
      " HAS BEEN UPDATED" );
  end if ;
  EXEC ADABAS
  FETCH VEHI
  END-EXEC
end loop ;

EXEC ADABAS
CLOSE VEHI
END-EXEC
EXEC ADABAS
COMMIT WORK
END-EXEC

else
  PUT_LINE ( "NO EMPLOYEES FOUND WITH PERSONNEL-ID " &
    PERSONNEL_NUMBER ) ;
end if ;
EXEC ADABAS
DBCLOSE
END-EXEC
end AEX2 ;
```

Example 3

```

with TYPES, ADABAS_GENERIC_CALLS, TEXT_IO ;
use  TYPES, TEXT_IO ;
-- SALARY INCREASE
-- THIS PROGRAM INCREASES THE SALARY OF EVERY EMPLOYEE BY
-- 4 PERCENT.
-- THE DEPARTMENT, THE OVERALL AMOUNT OF PAY RISE FOR THE
-- DEPARTMENT AND THE PAY RISE FOR ALL DEPARTMENTS WILL BE PRINTED
-- OUT.
-- THE PROGRAM IS RESTARTABLE. AFTER AN ABNORMAL TERMINATION THE
-- PROGRAM EXECUTION WOULD RESTART FROM THE LAST DEPARTMENT
-- WHOSE SALARY UPDATE HAD BEEN COMPLETED BEFORE THE ABEND
-- OCCURED.

procedure AEX3 is
  type COMMIT_DATA_1 is
    RECORD
      COMMIT_DEPARTMENT : STRING(1..6) := "      " ;
      COMMIT_SUM : INTEGER := 0 ;
      COMMIT_FIL : STRING(1..490) := (1..490 => ' ');
    end record ;
  COMMIT_DATA : COMMIT_DATA_1 ;
  COMMIT_DATA_2 : STRING(1..500);
  for COMMIT_DATA use at COMMIT_DATA_2'ADDRESS;
  START_DEPT : STRING(1..6) := "      " ;
  J : INTEGER := 0 ;
  NEW_SALARY : INTEGER := 0 ;
  INCREASE : INTEGER := 0 ;
  SUM_DEPARTMENT : INTEGER := 0 ;
  SUM_TOTAL : INTEGER := 0 ;
  FILLE1 : STRING(1..10) := " DEPARTMENT" ;
  FILLE2 : STRING(1..15) := (1..15 => ' ' ) ;
  FILLE3 : STRING(1..15) := "SALARY INCREASE " ;
  HEADER : STRING(1..40) := FILLE1 & FILLE2 & FILLE3 ;
  HEADER2 : STRING(1..40) := (1..40 => '*') ;
  SPACE_LINE : STRING(1..40) := (1..40 => ' ' ) ;

  EXEC ADABAS
BEGIN DECLARE SECTION
  END-EXEC

  EXEC ADABAS
  HISTOGRAM
  DECLARE EMP1 CURSOR FOR
  SELECT DEPT
  FROM EMPLOYEES EMPLOYEES_1
  WHERE DEPT GE :COMMIT_DATA.COMMIT_DEPARTMENT

```

```

GROUP BY DEPT
END-EXEC

EXEC ADABAS
READ LOGICAL
DECLARE EMP2 CURSOR FOR
SELECT PERSONNEL-ID, DEPT, SALARY, INCOME(COUNT)
FROM EMPLOYEES
WHERE DEPT GE :START_DEPT
ORDER BY DEPT
OPTIONS HOLD
END-EXEC

```

```

begin

EXEC ADABAS
CONNECT 'INCREASE'
UPD=EMPLOYEES
AND USERDATA INTO :COMMIT_DATA_2
END-EXEC
--
-- A HISTOGRAM STATEMENT IS USED TO ASCERTAIN THE NUMBER OF
-- EMPLOYEES PER DEPARTMENT
--

EXEC ADABAS
OPEN EMP1
END-EXEC

EXEC ADABAS
FETCH EMP1
END-EXEC

if COMMIT_DATA.COMMIT_DEPARTMENT /= " " then

PUT_LINE (" LAST PROGRAM RUN TERMINATED ABNORMALLY ");
PUT_LINE (" LAST DEPARTMENT WAS: " &
COMMIT_DATA.COMMIT_DEPARTMENT) ;

EXEC ADABAS
FETCH EMP1
END-EXEC

end if ;

START_DEPT := EMPLOYEES_1.DEPT ;

EXEC ADABAS
OPEN EMP2
END-EXEC

PUT_LINE(HEADER) ;

```

```

PUT_LINE(HEADER2) ;
PUT_LINE(SPACE_LINE) ;

while ADACODE /= 3 loop
--
-- THE EMPLOYEES FILE WILL BE READ UNTIL ALL RECORDS FOR THE
-- DEPARTMENT HAVE BEEN PROCESSED AND THE SALARY HAS BEEN
-- UPDATED.
--
  J := 1 ;
  while J <= EMPLOYEES_1.QUANTITY loop
    EXEC ADABAS
    FETCH EMP2
    END-EXEC
    J := J + 1 ;
-- THE SALARY INCREASE CAN BE EXECUTED WHEN THE COUNT OF THE
-- PERIODIC GROUP IS LESS THAN 40.
    if EMPLOYEES.C_INCOME < 40 then
      INCREASE := (EMPLOYEES.SALARY(1) * 4)/100 ;
      NEW_SALARY := EMPLOYEES.SALARY(1) + INCREASE ;
      EMPLOYEES.SALARY(2..40) := EMPLOYEES.SALARY(1..39) ;
      EMPLOYEES.SALARY(1) := NEW_SALARY ;
      EXEC ADABAS
      UPDATE EMPLOYEES
      WHERE CURRENT OF EMP2
      END-EXEC
      SUM_DEPARTMENT := SUM_DEPARTMENT + INCREASE ;
      SUM_TOTAL := SUM_TOTAL + INCREASE ;
    else
      PUT_LINE("UPDATE PERSON " & EMPLOYEES.PERSONNEL_ID &
        "NOT POSSIBLE") ;
    end if ;
  end loop ;
  PUT_LINE(" " & EMPLOYEES.DEPT & " " &
    INTEGER'IMAGE(SUM_DEPARTMENT)) ;
  SUM_DEPARTMENT := 0 ;
  COMMIT_DATA.COMMIT_DEPARTMENT := EMPLOYEES.DEPT ;
  COMMIT_DATA.COMMIT_SUM := SUM_TOTAL ;
  EXEC ADABAS
  COMMIT WORK
  USERDATA = :COMMIT_DATA_2
  END-EXEC

  EXEC ADABAS
  FETCH EMP1
  END-EXEC
end loop ;

EXEC ADABAS
CLOSE EMP1
END-EXEC
EXEC ADABAS

```

```
CLOSE EMP2
    END-EXEC
    PUT_LINE(SPACE_LINE) ;
    SPACE_LINE(1..50) := (1..50 => '-') ;
    PUT_LINE(SPACE_LINE) ;
    SPACE_LINE(1..50) := (1..50 => ' ') ;
    PUT_LINE(SPACE_LINE) ;
    PUT_LINE("TOTAL SALARY INCREASE : " & INTEGER'IMAGE(SUM_TOTAL)) ;
    COMMIT_DATA.COMMIT_DEPARTMENT := "      " ;

    EXEC ADABAS
    DBCLOSE
    USERDATA = :COMMIT_DATA_2
    END-EXEC
end AEX3 ;
```


12 APPENDIX E - EXAMPLE OF ADA CODE GENERATED BY ADABAS NATIVE SQL

```
with TYPES, ADABAS_GENERIC_CALLS, TEXT_IO ;
use  TYPES, TEXT_IO ;
--
-- AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH
-- CONTAINS FIELDS TAKEN FROM TWO FILES.  THE FIELDS PERSONNEL-ID
-- NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,
-- PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE
-- FOLLOWING CONDITION:
--   PERSONNEL-ID BETWEEN 10000001 AND 19999999
--   MODEL-VEAR-MAKE >
--   CLASS = 'C'

procedure AEX1 is
  START_MODEL : STRING (1..20) := "MERCEDES-BENZ      ";
  START_YEAR_MAKE : STRING (1..2) := "86" ;
  START_MODEL_YEAR_MAKE : STRING(1..22) := START_MODEL &
      START_YEAR_MAKE ;

  FILLE1 : STRING(1..20) := " PERSONNEL-ID          " ;
  FILLE2 : STRING(1..17) := " NAME                  " ;
  FILLE3 : STRING(1..18) := " FIRST-NAME           " ;
  FILLE4 : STRING(1..6) := "BIRTH      " ;
  FILLE5 : STRING(1..3) := "SEX" ;
  HEADER : STRING(1..64) := FILLE1 & FILLE2 & FILLE3 & FILLE4
      & FILLE5 ;
  HEADER2:  STRING(1..64) := (1..64 => '*');
  SPACE_LINE : STRING(1..80) := (1..80 => ' ');
--
-- EXEC ADABAS
-- BEGIN DECLARE SECTION
-- END-EXEC
--
```

```

ADACODE : SHORT_INTEGER := 0 ;
CB_OPN : CONTROL_BLOCK :=
    (FILLER1           => "AS"           ,
     COMMAND_CODE     => " "           ,
     COMMAND_ID       => "OPEN"        ,
     FILE_NUMBER      => 0             ,
     RESPONSE_CODE    => 0             ,
     ISN              => 0             ,
     ISN_LOWER_LIMIT  => 0             ,
     ISN_QUANTITY     => 0             ,
     FORMAT_BUFFER_LENGTH => 0         ,
     RECORD_BUFFER_LENGTH => 0         ,
     SEARCH_BUFFER_LENGTH => 0         ,
     VALUE_BUFFER_LENGTH => 0         ,
     ISN_BUFFER_LENGTH => 4           ,
     COMMAND_OPTION_1 => " "           ,
     COMMAND_OPTION_2 => " "           ,
     ADDITIONS_1      => " "           ,
     ADDITIONS_2      => " "           ,
     ADDITIONS_3      => " "           ,
     ADDITIONS_4      => " "           ,
     ADDITIONS_5      => " "           ,
     COMMAND_TIME     => 0             ,
     USER_AREA        => "AS "        ) ;

FORMAT_BUF_OPN      : FORMAT_BUFFER (1..0001) ;
SEARCH_BUF_OPN     : SEARCH_BUFFER (1..0001) ;
VB_OPN             : VALUE_BUFFER (1..0001) ;
RB_OPN             : RECORD_BUFFER (1..1500) ;
ISN_BUF_OPN        : ISN_BUFFER (1..0001) ;
package A_OPN is new ADABAS_GENERIC_CALLS
(FORMAT_BUFFER,RECORD_BUFFER,SEARCH_BUFFER,VALUE_BUFFER) ;
DDFILE : STRING(1..3) := "061" ;
CSEQ : STRING(1..8) ;
CLN1 : CLN_TYPE ;
CLN2 : CLN_TYPE ;
TRCE : STRING(1..7) ;
CLNUM : SHORT_INTEGER ;
SQLRSP : SHORT_INTEGER ;
SQLQTY : INTEGER ;
SQLISN : INTEGER ;
type FORMAT_BUFEMPL_1 is
record
    FILLE001 : STRING(1..32) := "AA,8,A,AE,20,A,AC,20,A,AH,6,U,AG" ;
    FILLE002 : STRING(1..05) := ",1,A." ;
end record ;
FORMAT_BUFEMPL : FORMAT_BUFEMPL_1 ;
type SEARCH_BUFEMPL_1 is
record
    FILLE001 : STRING(1..32) := "(22,AA,24,AC)/22/AA,8,A,S,AA,8,A" ;
    FILLE002 : STRING(1..27) := ",D,/24/A0,22,A,GT,D,AH,1,A." ;
end record ;

```



```

SEARCH_BUFEMPL          : SEARCH_BUFEMPL_1          ;
type RECORD_BUFEMPL     is
  record
    PERSONNEL_ID        : STRING (1..0008)          ;
    NAME                 : STRING (1..0020)          ;
    FIRST_NAME          : STRING (1..0020)          ;
    BIRTH               : STRING (1..0006)          ;
    SEX                 : STRING (1..0001)          ;
    ISN                 : INTEGER                   ;
    QUANTITY            : INTEGER                   ;
    RESPONSE_CODE       : SHORT_INTEGER             ;
  end record

```

```

EMPLOYEES               : RECORD_BUFEMPL           ;
type VALUE_BUFEMPL      is
  record
    V_PERSONNEL_ID_F    : STRING (1..0008)          ;
                        := (1..0008 => ' ' ) ;
    V_PERSONNEL_ID_T    : STRING (1..0008)          ;
                        := (1..0008 => ' ' ) ;
    V_MODEL_YEAR_MAKE   : STRING (1..0022)          ;
                        := (1..0022 => ' ' ) ;
    V_CLASS             : STRING (1..0001)          ;
                        := (1..0001 => ' ' ) ;
  end record
VBEMPL                  : VALUE_BUFEMPL           ;
ISN_BUFEMPL             : ISN_BUFFER (1..0001)    ;
package AEMPL is new ADABAS_GENERIC_CALLS
  (FORMAT_BUFEMPL_1, RECORD_BUFEMPL, SEARCH_BUFEMPL_1, VALUE_BUFEMPL) ;
CBEMPL : CONTROL_BLOCK :=
  (FILLER1              => "AS" ,
  COMMAND_CODE          => " " ,
  COMMAND_ID           => "EMPL" ,
  FILE_NUMBER          => 22 ,
  RESPONSE_CODE        => 0 ,
  ISN                  => 0 ,
  ISN_LOWER_LIMIT      => 0 ,
  ISN_QUANTITY         => 0 ,
  FORMAT_BUFFER_LENGTH => 37 ,
  RECORD_BUFFER_LENGTH => 55 ,
  SEARCH_BUFFER_LENGTH => 59 ,
  VALUE_BUFFER_LENGTH  => 39 ,
  ISN_BUFFER_LENGTH    => 4 ,
  COMMAND_OPTION_1     => " " ,
  COMMAND_OPTION_2     => " " ,
  ADDITIONS_1          => " " ,
  ADDITIONS_2          => " " ,
  ADDITIONS_3          => " " ,
  ADDITIONS_4          => " " ,
  ADDITIONS_5          => " " ,
  COMMAND_TIME         => 0 ,

```

```

                                USER_AREA          => "AS " ) ;
ISNSIZEEMPL                    : INTEGER          ;
ISNMOREEMPL                    : INTEGER          ;
ISNINDEEMPL                    : INTEGER          ;
EOFEMPL                        : BOOLEAN := FALSE ;

--
--      EXEC ADABAS
--  DECLARE EMPL CURSOR FOR
--  SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX
--  FROM EMPLOYEES, VEHICLES
--  WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
--        AND PERSONNEL-ID BETWEEN "10000001" AND "19999999"
--        AND VEHICLES.MODEL-YEAR-MAKE > :START_MODEL_YEAR_MAKE
--        AND VEHICLES.CLASS = "C"
--  END-EXEC

```

```

--
begin
--
--      EXEC ADABAS
--  OPEN EMPL
--  END-EXEC
--
--      VBEMPL.V_PERSONNEL_ID_F := "10000001" ;
--      VBEMPL.V_PERSONNEL_ID_T := "19999999" ;
--      VBEMPL.V_MODEL_YEAR_MAKE := START_MODEL_YEAR_MAKE ;
--      VBEMPL.V_CLASS := "C" ;
ISNSIZEEMPL := INTEGER(CBEMPL.ISN_BUFFER_LENGTH / 4) ;
ISNINDEEMPL := 1 ;
CBEMPL.ISN_LOWER_LIMIT := 0 ;
CBEMPL.COMMAND_OPTION_1 := " " ;
CBEMPL.COMMAND_OPTION_2 := " " ;
CBEMPL.ISN_QUANTITY := 0 ;
CBEMPL.ISN_BUFFER_LENGTH := 0 ;
CBEMPL.COMMAND_CODE := "S1" ;
AEMPL.ADABAS (
    CBEMPL,FORMAT_BUFEMPL,
    EMPLOYEES,SEARCH_BUFEMPL,VBEMPL,
    ISN_BUFEMPL
) ;
EMPLOYEES.RESPONSE_CODE :=
    CBEMPL.RESPONSE_CODE ;
EMPLOYEES.QUANTITY :=
    CBEMPL.ISN_QUANTITY ;
EMPLOYEES.ISN :=
    CBEMPL.ISN ;
if CBEMPL.RESPONSE_CODE /= 0
then
CSEQ := "00000000" ;

```

```

CLN1(01) := "      EXEC ADABAS      " ;
CLN2(01) := "                      " ;
CLN1(02) := "      OPEN EMPL      " ;
CLN2(02) := "                      " ;
CLN1(03) := "      END-EXEC      " ;
CLN2(03) := "                      " ;
CLNNUM := 03 ;
  AEMPL.RESPF
    (CBEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
     EMPLOYEES                      ,SEARCH_BUFEMPL,VBEMPL,
     CLN1,CLN2,TRCE,CLNNUM) ;
  end if ;
ISNMOREEMPL := CBEMPL.ISN_QUANTITY ;
if ISNMOREEMPL > 0 then
  EOFEMPL := FALSE ;
else
  EOFEMPL := TRUE ;
end if ;
if ISNMOREEMPL < ISNSIZEEMPL then
  ISNSIZEEMPL := ISNMOREEMPL ;
end if ;
ISNINDEMPL :=0 ;

PUT_LINE (HEADER) ;
PUT_LINE (HEADER2) ;
PUT_LINE (SPACE_LINE) ;

```

```

--
--      EXEC ADABAS
--      FETCH EMPL
--      END-EXEC
--
if ISNINDEMPL = ISNMOREEMPL then
  EOFEMPL := TRUE ;
end if ;
if not(EOFEMPL) then
  EOFEMPL := FALSE ;
  CBEMPL.COMMAND_OPTION_2 := "N" ;
  CBEMPL.COMMAND_OPTION_1 := " " ;
  CBEMPL.COMMAND_CODE := "L1" ;
  AEMPL.ADABAS (
    CBEMPL,FORMAT_BUFEMPL,
    EMPLOYEES                      ,SEARCH_BUFEMPL,VBEMPL,
    ISN_BUFEMPL                      ) ;
  EMPLOYEES.RESPONSE_CODE :=
  CBEMPL.RESPONSE_CODE ;
  EMPLOYEES.QUANTITY :=
  CBEMPL.ISN_QUANTITY ;
  EMPLOYEES.ISN :=
  CBEMPL.ISN ;
if CBEMPL.RESPONSE_CODE = 3 then

```

```

    EOFEMPL := TRUE ;
else
if CBEMPL.RESPONSE_CODE /= 0
    then
CSEQ := "00000000" ;
CLN1(01) := "          EXEC ADABAS          " ;
CLN2(01) := "          " ;
CLN1(02) := "          FETCH EMPL          " ;
CLN2(02) := "          " ;
CLN1(03) := "          END-EXEC          " ;
CLN2(03) := "          " ;
CLNNUM := 03 ;
    AEMPL.RESPF
        (CBEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
         EMPLOYEES,SEARCH_BUFEMPL,VBEMPL,
         CLN1,CLN2,TRCE,CLNNUM) ;
    end if ;
end if ;
if EOFEMPL then
    ADACODE := 003 ;
else
    ADACODE := 0 ;
end if ;

while ADACODE /= 3 loop
    PUT_LINE ( " " & EMPLOYEES.PERSONNEL_ID & " " & EMPLOYEES.NAME &
              " " & EMPLOYEES.FIRST_NAME & " " & EMPLOYEES.BIRTH & " "
              & EMPLOYEES.SEX ) ;

```

```

--
--      EXEC ADABAS
--      FETCH EMPL
--      END-EXEC
--
if ISNINDEMPL = ISNMOREEMPL then
    EOFEMPL := TRUE ;
end if ;
if not(EOFEMPL) then
EOFEMPL := FALSE ;
CBEMPL.COMMAND_OPTION_2 := "N" ;
CBEMPL.COMMAND_OPTION_1 := " " ;
CBEMPL.COMMAND_CODE := "L1" ;
AEMPL.ADABAS (
    CBEMPL,FORMAT_BUFEMPL,
    EMPLOYEES,SEARCH_BUFEMPL,VBEMPL,
    ISN_BUFEMPL ) ;
EMPLOYEES.RESPONSE_CODE :=
    CBEMPL.RESPONSE_CODE ;
EMPLOYEES.QUANTITY :=
    CBEMPL.ISN_QUANTITY ;

```

```

EMPLOYEES.ISN                               :=
                                         CBEMPL.ISN ;
if CBEMPL.RESPONSE_CODE = 3 then
  EOFEMPL := TRUE ;
else
  if CBEMPL.RESPONSE_CODE /= 0
    then
      CSEQ := "00000000" ;
      CLN1(01) := "          EXEC ADABAS          " ;
      CLN2(01) := "          " ;
      CLN1(02) := "          FETCH EMPL          " ;
      CLN2(02) := "          " ;
      CLN1(03) := "          END-EXEC          " ;
      CLN2(03) := "          " ;
      CLNNUM := 03 ;
      AEMPL.RESPF
        (CBEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
         EMPLOYEES                               ,SEARCH_BUFEMPL,VBEMPL,
         CLN1,CLN2,TRCE,CLNNUM) ;
    end if ;
  end if ;
  end if ;
  if EOFEMPL then
    ADACODE := 003 ;
  else
    ADACODE := 0 ;
  end if ;
end loop ;

```

```

--
--      EXEC ADABAS
--      CLOSE EMPL
--      END-EXEC
--
CBEMPL.COMMAND_OPTION_1 := "I" ;
CBEMPL.COMMAND_OPTION_2 := "S" ;
CBEMPL.COMMAND_CODE := "RC" ;
AEMPL.ADABAS (
  CBEMPL,FORMAT_BUFEMPL,
  EMPLOYEES                               ,SEARCH_BUFEMPL,VBEMPL,
  ISN_BUFEMPL                               ) ;
EMPLOYEES.RESPONSE_CODE :=
  CBEMPL.RESPONSE_CODE ;
EMPLOYEES.QUANTITY :=
  CBEMPL.ISN_QUANTITY ;
EMPLOYEES.ISN :=
  CBEMPL.ISN ;
if CBEMPL.RESPONSE_CODE /= 0
  then
    CSEQ := "00000000" ;

```

```

CLN1(01) := "          EXEC ADABAS          " ;
CLN2(01) := "          " ;
CLN1(02) := "      CLOSE EMPL          " ;
CLN2(02) := "          " ;
CLN1(03) := "          END-EXEC          " ;
CLN2(03) := "          " ;
CLNNUM := 03 ;
    AEMPL.RESPF
      (CBEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
        EMPLOYEES          ,SEARCH_BUFEMPL,VBEMPL,
        CLN1,CLN2,TRCE,CLNNUM) ;
    end if ;

--
--      EXEC ADABAS
-- DBCLOSE
--      END-EXEC
--

CB_OPN.RECORD_BUFFER_LENGTH := 1500 ;
CB_OPN.COMMAND_OPTION_2 := " " ;
CB_OPN.COMMAND_CODE := "CL" ;
A_OPN.ADABAS (
    CB_OPN,FORMAT_BUF_OPN,
    RB_OPN          ,SEARCH_BUF_OPN,VB_OPN,
    ISN_BUF_OPN          ) ;
if CB_OPN.RESPONSE_CODE /= 0
    then
CSEQ := "00000000" ;
CLN1(01) := "          EXEC ADABAS          " ;
CLN2(01) := "          " ;
CLN1(02) := "      DBCLOSE          " ;
CLN2(02) := "          " ;
CLN1(03) := "          END-EXEC          " ;
CLN2(03) := "          " ;
CLNNUM := 03 ;
    A_OPN.RESPF
      (CB_OPN,DDFILE,CSEQ,FORMAT_BUF_OPN,
        RB_OPN          ,SEARCH_BUF_OPN,VB_OPN,
        CLN1,CLN2,TRCE,CLNNUM) ;
    end if ;
end AEX1 ;

```

13 APPENDIX F - COBOL EXAMPLES

▪ Example 1	288
▪ Example 2	290
▪ Example 3	292

This chapter covers the following topics:

Example 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CEX1.
REMARKS.
* AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH      *
* CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS PERSONNEL-ID *
* NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,          *
* PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE        *
* FOLLOWING CONDITION:
*     PERSONNEL-ID BETWEEN 10000001 AND 19999999
*     MODEL-YEAR-MAKE >
*     CLASS = 'C'
ENVIRONMENT DIVISION.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 START-MODEL-YEAR-MAKE.
   02 START-MODEL          PIC X(20) VALUE 'MERCEDES-BENZ'.
   02 START-YEAR-MAKE     PIC 9(2)  VALUE 86.
*
01 HEADER.
   02 FILLER              PIC X(12) VALUE 'PERSONNEL-ID'.
   02 FILLER              PIC X(8)  VALUE SPACE.
   02 FILLER              PIC X(4)  VALUE 'NAME'.
   02 FILLER              PIC X(13) VALUE SPACE.
   02 FILLER              PIC X(10) VALUE 'FIRST NAME'.
   02 FILLER              PIC X(8)  VALUE SPACE.
   02 FILLER              PIC X(5)  VALUE 'BIRTH'.
   02 FILLER              PIC X(1)  VALUE SPACE.
   02 FILLER              PIC X(3)  VALUE 'SEX'.
01 HEADER2                PIC X(64) VALUE ALL '*'.
01 SPACE-LINE             PIC X(80) VALUE SPACE.
01 LINE1.
   02 FILLER              PIC X(2)  VALUE SPACE.
   02 PERSONNEL-NR        PIC X(8)  VALUE SPACE.
   02 FILLER              PIC X(3)  VALUE SPACE.
   02 LAST-NAME           PIC X(20) VALUE SPACE.
   02 FILLER              PIC X(1)  VALUE SPACE.
   02 F-NAME              PIC X(20) VALUE SPACE.
   02 FILLER              PIC X(1)  VALUE SPACE.
   02 BIRTHDAY           PIC X(6)  VALUE SPACE.
   02 FILLER              PIC X(1)  VALUE SPACE.
   02 KIND                PIC X(1)  VALUE SPACE.
```



```

*
      EXEC ADABAS
      BEGIN DECLARE SECTION
      END-EXEC
*
      EXEC ADABAS
      DECLARE EMPL CURSOR FOR
      SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX
      FROM EMPLOYEES, VEHICLES
      WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
           AND PERSONNEL-ID BETWEEN '10000001' AND '19999999'
           AND VEHICLES.MODEL-YEAR-MAKE > :START-MODEL-YEAR-MAKE
           AND VEHICLES.CLASS = 'C'
      END-EXEC

```

```

PROCEDURE DIVISION.
*
      DISPLAY HEADER.
      DISPLAY HEADER2.
      DISPLAY SPACE-LINE.
*
      EXEC ADABAS
      OPEN EMPL
      END-EXEC
*
      EXEC ADABAS
      FETCH EMPL
      END-EXEC
*
      PERFORM READ-EMPLOYEES UNTIL ADACODE = 3.
*
      EXEC ADABAS
      CLOSE EMPL
      END-EXEC
*
      EXEC ADABAS
      DBCLOSE
      END-EXEC
*
      STOP RUN.
*
      READ-EMPLOYEES.
      MOVE PERSONNEL-ID TO PERSONNEL-NR.
      MOVE NAME TO LAST-NAME.
      MOVE FIRST-NAME TO F-NAME.
      MOVE BIRTH TO BIRTHDAY.
      MOVE SEX TO KIND.
      DISPLAY LINE1.
      MOVE SPACE TO LINE1.
*
      EXEC ADABAS

```

```

FETCH EMPL
END-EXEC

```

Example 2

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CEX2.
REMARKS.
* DELETE AN EMPLOYEE RECORD AND RELEASE ALL CARS WHICH ARE      *
* ASSIGNED TO THIS EMPLOYEE. A PRIVATE CAR WILL BE DELETED      *
* AND A COMPANY CAR WILL BE MADE A POOL-CAR WHICH IS IDENTIFIED  *
* BY ITS PERSONNEL-ID CONTAINING ONLY THE COUNTRY CODE.         *
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
01  PERSONNEL-NUMBER          PIC X(8)  VALUE '20007100'.
01  EMPLOYEE-ISN             PIC 9(9)  COMP VALUE ZERO.
01  COUNTRY-NUMBER.
    02  COUNTRY-NO           PIC X(1)  VALUE SPACE.
    02  FILLER                PIC X(14) VALUE SPACE.
*
    EXEC ADABAS
    BEGIN DECLARE SECTION
    END-EXEC
*
    EXEC ADABAS
    READ LOGICAL
    DECLARE VEHI CURSOR FOR
    SELECT REG-NUM, PERSONNEL-ID, CLASS
    FROM VEHICLES
    WHERE PERSONNEL-ID GE :PERSONNEL-NUMBER
    OPTIONS HOLD
    ORDER BY PERSONNEL-ID
    END-EXEC
*
PROCEDURE DIVISION.
*
*** FIND EMPLOYEE
*
    EXEC ADABAS
    FIND
    SELECT
    FROM EMPLOYEES EMPLOYEES-1
    WHERE PERSONNEL-ID = :PERSONNEL-NUMBER
    OPTIONS HOLD
    END-EXEC
*

```

```

*** IF THE PERSONNEL-ID EXISTS DELETE THE EMPLOYEE AND READ THE
*** VEHICLES FILE
*
  IF QUANTITY OF EMPLOYEES-1 = 1
    MOVE ISN OF EMPLOYEES-1 TO EMPLOYEE-ISN
    PERFORM DELETE-EMPLOYEE
    PERFORM READ-VEHICLES-FILE
  ELSE
    DISPLAY
      'NO EMPLOYEE FOUND WITH PERSONNEL-ID ', PERSONNEL-NUMBER.
*
      EXEC ADABAS
DBCLOSE
      END-EXEC
*
  STOP RUN.

```

```

*
DELETE-EMPLOYEE.
  EXEC ADABAS
  DELETE
  FROM EMPLOYEES
  WHERE ISN = :EMPLOYEE-ISN
  END-EXEC
*
  DISPLAY 'EMPLOYEE ', PERSONNEL-NUMBER, ' HAS BEEN DELETED'.
*
READ-VEHICLES-FILE.
  EXEC ADABAS
  OPEN VEH1
  END-EXEC
*
  EXEC ADABAS
  FETCH VEH1
  END-EXEC
*
  PERFORM READ-VEHICLES UNTIL ADACODE = 3 OR
    PERSONNEL-ID OF VEHICLES > PERSONNEL-NUMBER.
*
  EXEC ADABAS
  CLOSE VEH1
  END-EXEC
*
  EXEC ADABAS
  COMMIT WORK
  END-EXEC
*
READ-VEHICLES.
  IF CLASS = 'P'
    PERFORM DELETE-PRIVATE-CAR
  ELSE

```

```
        PERFORM UPDATE-COMPANY-CAR.
*
        EXEC ADABAS
        FETCH VEHI
        END-EXEC
*
        DELETE-PRIVATE-CAR.
        EXEC ADABAS
        DELETE
        FROM VEHICLES
        WHERE CURRENT OF VEHI
        END-EXEC
        DISPLAY 'PRIVATE CAR ', REG-NUM, ' HAS BEEN DELETED'.
*
        UPDATE-COMPANY-CAR.
        MOVE PERSONNEL-ID OF VEHICLES TO COUNTRY-NUMBER.
        MOVE COUNTRY-NO TO PERSONNEL-ID OF VEHICLES.
*
        EXEC ADABAS
        UPDATE VEHICLES
        WHERE CURRENT OF VEHI
        END-EXEC
        DISPLAY 'COMPANY CAR ', REG-NUM, ' HAS BEEN UPDATED'.
```

Example 3

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CEX3.
REMARKS.
* SALARY INCREASE.
* THIS PROGRAM INCREASES THE SALARY OF EVERY EMPLOYEE BY
* 4 PERCENT.
* THE DEPARTMENT, THE OVERALL AMOUNT OF PAY RISE FOR THE
* DEPARTMENT AND THE PAY RISE FOR ALL DEPARTMENTS WILL BE PRINTED
* OUT.
* THE PROGRAM IS RESTARTABLE. AFTER AN ABNORMAL TERMINATION THE
* PROGRAM EXECUTION WOULD RESTART WITH THE LAST DEPARTMENT
* WHOSE SALARY UPDATE HAD BEEN COMPLETED BEFORE THE ABEND
* OCCURED.
```

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
01 COMMIT-DATA.
   02 COMMIT-DEPARTMENT    PIC X(6)          VALUE SPACE.
   02 COMMIT-SUM           PIC S9(10) COMP-3  VALUE +0.
01 START-DEPT             PIC X(6)          VALUE SPACE.
01 IND                    PIC 9(4)         COMP   VALUE 0.
01 I                      PIC 9(4)         COMP   VALUE 0.
01 J                      PIC 9(4)         COMP   VALUE 0.
01 NEW-SALARY             PIC S9(9)        COMP-3  VALUE +0.
01 INCREASE               PIC S9(9)        COMP-3  VALUE +0.
01 SUM-DEPARTMENT        PIC S9(10) COMP-3  VALUE +0.
01 SUM-TOTAL              PIC S9(11) COMP-3  VALUE +0.
*
01 HEADER.
   02 FILLER                PIC X(10) VALUE 'DEPARTMENT'.
   02 FILLER                PIC X(15) VALUE SPACE.
   02 FILLER                PIC X(15) VALUE 'SALARY INCREASE'.
01 HEADER2                PIC X(40) VALUE ALL '*'.
01 SPACE-LINE             PIC X(50) VALUE SPACE.
01 LINE1.
   02 FILLER                PIC X(3)  VALUE SPACE.
   02 DEPARTMENT           PIC X(6)  VALUE SPACE.
   02 FILLER                PIC X(16) VALUE SPACE.
   02 SUM-DEPT             PIC Z,ZZZ,ZZZ,ZZ9.
01 LAST-LINE.
   02 FILLER                PIC X(21) VALUE 'TOTAL SALARY INCREASE'.
   02 FILLER                PIC X(3)  VALUE ' : '.
   02 TOTAL-SUM-DEPT      PIC ZZ,ZZZ,ZZZ,ZZZ.
*
      EXEC ADABAS
      BEGIN DECLARE SECTION
      END-EXEC
*
      EXEC ADABAS
      HISTOGRAM
      DECLARE EMP1 CURSOR FOR
      SELECT DEPT
      FROM EMPLOYEES EMPLOYEES-1
      WHERE DEPT GE :COMMIT-DEPARTMENT
      GROUP BY DEPT
      END-EXEC
*
      EXEC ADABAS
      READ LOGICAL
      DECLARE EMP2 CURSOR FOR
      SELECT PERSONNEL-ID, DEPT, SALARY, INCOME(COUNT)
      FROM EMPLOYEES
      WHERE DEPT GE :START-DEPT

```

```
OPTIONS HOLD
ORDER BY DEPT
END-EXEC
```

*

```
PROCEDURE DIVISION.
```

*

```
EXEC ADABAS
CONNECT 'INCREASE'
UPD=EMPLOYEES
AND USERDATA INTO :COMMIT-DATA
END-EXEC
```

*

```
*** A HISTOGRAM STATEMENT IS USED TO ASCERTAIN THE NUMBER OF
*** EMPLOYEES PER DEPARTMENT
```

*

```
EXEC ADABAS
OPEN EMP1
END-EXEC
```

*

```
EXEC ADABAS
FETCH EMP1
END-EXEC
```

*

```
IF COMMIT-DATA NOT = ' '
PERFORM RESTART.
```

*

```
MOVE DEPT OF EMPLOYEES-1 TO START-DEPT.
```

*

```
EXEC ADABAS
OPEN EMP2
END-EXEC
```

*

```
DISPLAY HEADER.
DISPLAY HEADER2.
DISPLAY SPACE-LINE.
PERFORM HIST-EMPL UNTIL ADACODE = 3.
```

*

```
EXEC ADABAS
CLOSE EMP1
END-EXEC
```

*

```
EXEC ADABAS
CLOSE EMP2
END-EXEC
```

*

```
DISPLAY SPACE-LINE.
MOVE ALL '-' TO SPACE-LINE.
DISPLAY SPACE-LINE.
MOVE SPACES TO SPACE-LINE.
DISPLAY SPACE-LINE.
```

```

MOVE SUM-TOTAL TO TOTAL-SUM-DEPT.
DISPLAY LAST-LINE.
MOVE ' ' TO COMMIT-DATA.
*
      EXEC ADABAS
      DBCLOSE
      USERDATA = :COMMIT-DATA
      END-EXEC
*
      STOP RUN.
*
```

```

RESTART.
      DISPLAY 'LAST PROGRAM RUN TERMINATED ABNORMALLY'.
      DISPLAY 'LAST DEPARTMENT WAS: ', COMMIT-DEPARTMENT.
*
      EXEC ADABAS
      FETCH EMP1
      END-EXEC.
*
      HIST-EMPL.
*
*** THE EMPLOYEES FILE WILL BE READ UNTIL ALL RECORDS FOR THE
*** DEPARTMENT HAVE BEEN PROCESSED AND THE SALARY HAS BEEN
*** UPDATED.
*
      PERFORM READ-EMPL VARYING J FROM 1 BY 1 UNTIL
                                J > QUANTITY OF EMPLOYEES-1.
      MOVE DEPT OF EMPLOYEES TO DEPARTMENT.
      MOVE SUM-DEPARTMENT TO SUM-DEPT.
      MOVE ZERO TO SUM-DEPARTMENT.
      DISPLAY LINE1.
      MOVE SPACE TO LINE1.
*
      MOVE DEPT OF EMPLOYEES TO COMMIT-DEPARTMENT.
      MOVE SUM-TOTAL TO COMMIT-SUM.
      EXEC ADABAS
      COMMIT WORK
      USERDATA = :COMMIT-DATA
      END-EXEC
*
      EXEC ADABAS
      FETCH EMP1
      END-EXEC.
*
      READ-EMPL.
      EXEC ADABAS
      FETCH EMP2
      END-EXEC.
```

```
*
*** THE SALARY INCREASE CAN BE EXECUTED WHEN THE COUNT OF THE
*** PERIODIC GROUP IS LESS THAN 40.
*
  IF C-INCOME < 40
    PERFORM SALARY-INCREASE
  ELSE
    DISPLAY 'UPDATE PERSON ', PERSONNEL-ID, ' NOT POSSIBLE'.
*
SALARY-INCREASE.
  COMPUTE INCREASE = SALARY(1) * 0.04.
  COMPUTE NEW-SALARY = SALARY(1) + INCREASE.
  ADD 1 C-INCOME OF EMPLOYEES GIVING IND.
  PERFORM INCREASE-IN-SALARY VARYING I FROM C-INCOME BY -1
                                UNTIL I = 0.
  MOVE NEW-SALARY TO SALARY(1).
*
  EXEC ADABAS

  UPDATE EMPLOYEES
  WHERE CURRENT OF EMP2
  END-EXEC
*
  COMPUTE SUM-DEPARTMENT = SUM-DEPARTMENT + INCREASE.
  COMPUTE SUM-TOTAL = SUM-TOTAL + INCREASE.
*
INCREASE-IN-SALARY.
  MOVE SALARY(I) TO SALARY(IND).
  SUBTRACT 1 FROM IND.
```


14 APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```
1
0 000001          IDENTIFICATION DIVISION.
      00000010
000002          PROGRAM-ID. CEX1.
      00000020
000003          * WITH COBOL II SET THE NEXT LINE TO COMMENT
000004          *REMARKS.
000005          * AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH
      *
000006          * CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS
PERSONNEL-ID *
000007          * NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,
      *
000008          * PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE
      *
000009          * FOLLOWING CONDITION:
000010          *     PERSONNEL-ID BETWEEN 10000001 AND 19999999
000011          *     MODEL-YEAR-MAKE >
000012          ENVIRONMENT DIVISION.
      00000030
000013          DATA DIVISION.
      00000040
000014          WORKING-STORAGE SECTION.
      00000050
000015          01  START-MODEL-YEAR-MAKE.
      00000060
000016          02  START-MODEL          PIC X(20) VALUE
'MERCEDES-BENZ'. 00000070
000017          02  START-YEAR-MAKE     PIC 9(2)  VALUE 86.
      00000080
000018          *
000019          01  HEADER.
```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

0000090
000020          02 FILLER          PIC X(12) VALUE 'PERSONNEL-ID'.  ↵
0000100
000021          02 FILLER          PIC X(8)  VALUE SPACE.         ↵
0000110  IMP
000022          02 FILLER          PIC X(4)  VALUE 'NAME'.         ↵
0000120
000023          02 FILLER          PIC X(13) VALUE SPACE.         ↵
0000130  IMP
000024          02 FILLER          PIC X(10) VALUE 'FIRST NAME'.   ↵
0000140
000025          02 FILLER          PIC X(8)  VALUE SPACE.         ↵
0000150  IMP
000026          02 FILLER          PIC X(3)  VALUE 'SEX'.          ↵
0000160
000027          01 HEADER2         PIC X(64) VALUE ALL '*'.       ↵
0000170
000028          01 SPACE-LINE      PIC X(80) VALUE SPACE.         ↵
0000180  IMP
000029          01 LINE1.         ↵
0000190
000030          02 FILLER          PIC X(2)  VALUE SPACE.         ↵
0000200  IMP
000031          02 PERSONNEL-NR     PIC X(8)  VALUE SPACE.         ↵
0000210  IMP
000032          02 FILLER          PIC X(3)  VALUE SPACE.         ↵
0000220  IMP
000033          02 LAST-NAME        PIC X(20) VALUE SPACE.         ↵
0000230  IMP
000034          02 FILLER          PIC X(1)  VALUE SPACE.         ↵
0000240  IMP
000035          02 F-NAME           PIC X(20) VALUE SPACE.         ↵
0000250  IMP
000036          02 FILLER          PIC X(1)  VALUE SPACE.         ↵
0000260  IMP
000037          02 KIND            PIC X(1)  VALUE SPACE.         ↵
0000270  IMP
000038          *

```

```

000039          *
000040          *          EXEC ADABAS
000041          *          BEGIN DECLARE SECTION
000042          *          END-EXEC
000043          *
000044          01 ADACODE PIC 9(4) COMP VALUE 0.         ↵
ADABAS
000045          01 CONTROL-BLOCKOPN.         ↵
ADABAS
000046          03 FILLER10PN          PIC 9(4) COMP  VALUE 0.         ↵
ADABAS
000047          03 FILLER1-CHAROPN     REDEFINES FILLER10PN PIC XX.     ↵

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS 46
000048      03 COMMAND-CODEOPN          PIC XX          VALUE ←
SPACE.      ADABAS  IMP
000049      03 COMMAND-IDOPN           PIC X(4)         VALUE ←
'OPEN'.     ADABAS
000050      03 FILE-NUMBEROPN          PIC 9(4) COMP    VALUE ←
0.          ADABAS
000051      03 FILLER REDEFINES FILE-NUMBEROPN .           ←
ADABAS 50
000052      04 DBIDOPN PIC X.           ←
ADABAS
000053      04 FILLER PIC X.           ←
ADABAS
000054      03 RESPONSE-CODEOPN        PIC 9(4) COMP    VALUE 0. ←
ADABAS
000055      03 ISNOPN                  PIC 9(9) COMP    VALUE 0. ←
ADABAS
000056      03 ISN-LOWER-LIMITOPN     PIC 9(9) COMP    VALUE 0. ←
ADABAS
000057      03 ISN-QUANTITYOPN        PIC 9(9) COMP    VALUE 0. ←
ADABAS
0 000058      03 FORMAT-BUFFER-LENGTHOPN PIC 9(4) COMP    VALUE ←
0.          ADABAS
000059      03 FBL-CHAROPN REDEFINES FORMAT-BUFFER-LENGTHOPN PIC ←
XX.         ADABAS 58
000060      03 RECORD-BUFFER-LENGTHOPN PIC 9(4) COMP    VALUE ←
0.          ADABAS
000061      03 RBL-CHAROPN REDEFINES RECORD-BUFFER-LENGTHOPN PIC ←
XX.         ADABAS 60
000062      03 SEARCH-BUFFER-LENGTHOPN PIC 9(4) COMP    VALUE ←
0.          ADABAS
000063      03 VALUE-BUFFER-LENGTHOPN  PIC 9(4) COMP    VALUE ←
0.          ADABAS
000064      03 ISN-BUFFER-LENGTHOPN    PIC 9(4) COMP    VALUE ←
4.          ADABAS
000065      03 COMMAND-OPTION-10PN     PIC X            VALUE ←
SPACE.      ADABAS  IMP
000066      03 COMMAND-OPTION-20PN     PIC X            VALUE ←
SPACE.      ADABAS  IMP
000067      03 ADDITIONS-10PN          VALUE ←
SPACE.      ADABAS  IMP
000068      04 ADDITIONS-1-120PN PIC XX.           ←
ADABAS
000069      04 FILLER PIC XX.           ←
ADABAS
000070      04 ADDITIONS-1-580PN PIC X(4).         ←
ADABAS
000071      03 FILLER REDEFINES ADDITIONS-10PN .           ←
ADABAS 67
000072      04 ADDITIONS-1-BNOPN PIC 9(4) COMP.     ←
ADABAS
000073      04 FILLER PIC X(6).         ←

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS
000074      03 ADDITIONS-20PN          PIC X(4)      VALUE ←
SPACE.      ADABAS      IMP
000075      03 ADDITIONS-30PN          PIC X(8)      VALUE ←
SPACE.      ADABAS      IMP
000076      03 ADDITIONS-40PN          .              ←
ADABAS
000077      04 ADDITIONS-4-120PN PIC 9(4) COMP VALUE 0. ←
ADABAS
000078      04 ADDITIONS-4-340PN PIC 9(4) COMP VALUE 0. ←
ADABAS
000079      04 ADDITIONS-4-560PN PIC 9(4) COMP VALUE 0. ←
ADABAS
000080      04 ADDITIONS-4-780PN PIC 9(4) COMP VALUE 0. ←
ADABAS
000081      03 ADDITIONS-50PN          .              ←
ADABAS
000082      04 ADDITIONS-5-BNOPN  PIC 9(9) COMP VALUE 0. ←
ADABAS
000083      04 ADDITIONS-5-580PN  PIC X(4) VALUE SPACE. ←
ADABAS      IMP
000084      03 FILLER REDEFINES ADDITIONS-50PN .      ←
ADABAS      81
000085      04 ADDITIONS-5-10PN  PIC X.              ←
ADABAS
000086      04 ADDITIONS-5-280PN  PIC X(7).          ←
ADABAS
000087      03 COMMAND-TIMEOPN        PIC 9(9) COMP. ←
ADABAS
000088      03 USER-AREAOPN          PIC X(4)      VALUE ' ←
'.      ADABAS
000089      01  FORMAT-BUFOPN          PIC X.              ←
ADABAS
000090      01  SEARCH-BUFOPN         PIC X.              ←
ADABAS
000091      01  VALUE-BUFOPN          PIC X.              ←
ADABAS
000092      01  ISN-BUFOPN           PIC X.              ←
ADABAS
000093      01  OPENTYPE              PIC X(00010). ←
ADABAS
000094      01  RECORD-BUFOPN.        ←
ADABAS
000095      02  RECORD-BUFOPN-01      PIC X(00100). ←
ADABAS
000096      02  RECORD-BUFOPN-02      PIC X(00100). ←
ADABAS
000097      02  RECORD-BUFOPN-03      PIC X(00100). ←
ADABAS
000098      02  RECORD-BUFOPN-04      PIC X(00100). ←
ADABAS
000099      02  RECORD-BUFOPN-05      PIC X(00100). ←

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS
000100          02 RECORD-BUFOPN-06          PIC X(00100). ↵
ADABAS
000101          02 RECORD-BUFOPN-07          PIC X(00100). ↵
ADABAS
000102          02 RECORD-BUFOPN-08          PIC X(00100). ↵
ADABAS
000103          02 RECORD-BUFOPN-09          PIC X(00100). ↵
ADABAS
000104          02 RECORD-BUFOPN-10          PIC X(00100). ↵
ADABAS
000105          02 RECORD-BUFOPN-11          PIC X(00100). ↵
ADABAS
000106          02 RECORD-BUFOPN-12          PIC X(00100). ↵
ADABAS
000107          02 RECORD-BUFOPN-13          PIC X(00100). ↵
ADABAS
000108          02 RECORD-BUFOPN-14          PIC X(00100). ↵
ADABAS
000109          02 RECORD-BUFOPN-15          PIC X(00100). ↵
ADABAS
000110          01 DDFILE                      PIC 99999      VALUE ↵
7. ADABAS
000111          01 DDDBDID                    PIC 99999      VALUE ↵
11177. ADABAS
000112          01 CSEQ                      PIC X(8). ↵
ADABAS
000113          01 CLN1. ↵
ADABAS
000114          02 CLN1V PIC X(40) OCCURS 20. ↵
ADABAS

```

```

0 000115          01 CLN2. ↵
ADABAS
000116          02 CLN2V PIC X(40) OCCURS 20. ↵
ADABAS
000117          01 CLNNUM                    PIC 9(4) COMP. ↵
ADABAS
000118          01 TRCE                      PIC X(7). ↵
ADABAS
000119          01 SQLRSP                    PIC 9(4) COMP. ↵
ADABAS
000120          01 SQLQTY                    PIC 9(9) COMP. ↵
ADABAS
000121          01 SQLISN                    PIC 9(9) COMP. ↵
ADABAS
000122          01 ADA-FULL-INTOPN PIC 9(9) COMP VALUE 12288. ↵
ADABAS
000123          01 FILLER REDEFINES ADA-FULL-INTOPN. ↵
ADABAS 122
000124          02 FILLER PIC XX. ↵

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS
000125          02 ADA-HALF-INTOPN PIC XX.           ↵
ADABAS
000126          01 SAVE-DBID-10PN          PIC 9(9) COMP   VALUE 0. ↵
ADABAS
000127          01 SAVE-DBID-DEFOPN  REDEFINES SAVE-DBID-10PN .   ↵
ADABAS  126
000128          02 FILLER PIC X(2).             ↵
ADABAS
000129          02 SAVE-DBIDOPN  PIC 9(4) COMP.     ↵
ADABAS
000130          01 FORMAT-BUFEMPL.             ↵
ADABAS
000131          02 FILLER PIC X(30) VALUE        ↵
ADABAS
000132          'AA,8,A,AE,20,A,AC,20,A,AG,1,A.' . ↵
ADABAS
000133          01 SEARCH-BUFEMPL.             ↵
ADABAS
000134          02 FILLER PIC X(46) VALUE        ↵
ADABAS
000135          '(1,AA,2,AC)/1/AA,8,A,S,AA,8,A,D,/2/A0,24,A,GT.' . ↵
ADABAS
000136          01 EMPLOYEES.                  ↵
ADADATA
000137          02 RECORD-BUFEMPL.             ↵
ADADATA
000138          03 PERSONNEL-ID                PIC X(00008). ↵
ADADATA
000139          03 NAME                        PIC X(00020). ↵
ADADATA
000140          03 FIRST-NAME                 PIC X(00020). ↵
ADADATA
000141          03 SEX                        PIC X(00001). ↵
ADADATA
000142          02 ISN                         PIC 9(9) COMP ↵
VALUE 0.  ADADATA
000143          02 QUANTITY                    PIC 9(9) COMP ↵
VALUE 0.  ADADATA
000144          02 RESPONSE-CODE              PIC 9(4) COMP ↵
VALUE 0.  ADADATA
000145          01 VALUE-BUFEMPL.             ↵
ADABAS
000146          02 V-PERSONNEL-ID-F          PIC X(00008) ↵
ADABAS
000147          VALUE ↵
LOW-VALUE.  ADABAS  IMP
000148          02 V-PERSONNEL-ID-T          PIC X(00008) ↵
ADABAS
000149          VALUE ↵
LOW-VALUE.  ADABAS  IMP
000150          02 V-MODEL-YEAR-MAKE.         ↵

```

```

ADABAS
000151          03 S-YEAR                      PIC 9(0004)  ←
ADABAS
000152                                VALUE ←
0. ADABAS
000153          03 S-MAKE                      PIC X(00020) ←
ADABAS
000154                                VALUE ←
LOW-VALUE.     ADABAS  IMP
000155          01 ISN-BUFEMPL.                ←
ADABAS
000156          03 ISN-BUFVECEMPL OCCURS      1  PIC 9(9) COMP.  ←
ADABAS
000157          01 CONTROL-BLOCKEMPL.         ←
ADABAS
000158          03 FILLER1EMPL                PIC 9(4) COMP  VALUE 0. ←
ADABAS
000159          03 FILLER1-CHAREMPL REDEFINES  FILLER1EMPL PIC XX. ←
ADABAS  158
000160          03 COMMAND-CODEEMPL          PIC XX          VALUE ←
SPACE.         ADABAS  IMP
000161          03 COMMAND-IDEMPL            PIC X(4)          VALUE ←
'EMPL'.       ADABAS
000162          03 FILE-NUMBEREMPL          PIC 9(4) COMP  VALUE ←
1. ADABAS
000163          03 FILLER REDEFINES FILE-NUMBEREMPL. ←
ADABAS  162
000164          04 DBIDEMPL PIC X.           ←
ADABAS
000165          04 FILLER PIC X.             ←
ADABAS
000166          03 RESPONSE-CODEEMPL        PIC 9(4) COMP  VALUE 0. ←
ADABAS
000167          03 ISNEMPL                  PIC 9(9) COMP  VALUE 0. ←
ADABAS
000168          03 ISN-LOWER-LIMITEMPL      PIC 9(9) COMP  VALUE 0. ←
ADABAS
000169          03 ISN-QUANTITYEMPL        PIC 9(9) COMP  VALUE 0. ←
ADABAS
000170          03 FORMAT-BUFFER-LENGTHEMPL PIC 9(4) COMP  VALUE ←
30. ADABAS
000171          03 FBL-CHAREMPL REDEFINES  FORMAT-BUFFER-LENGTHEMPL PIC ←
XX. ADABAS  170
0 000172          03 RECORD-BUFFER-LENGTHEMPL PIC 9(4) COMP  VALUE ←
49. ADABAS
000173          03 RBL-CHAREMPL REDEFINES  RECORD-BUFFER-LENGTHEMPL PIC ←
XX. ADABAS  172
000174          03 SEARCH-BUFFER-LENGTHEMPL PIC 9(4) COMP  VALUE ←
46. ADABAS
000175          03 VALUE-BUFFER-LENGTHEMPL  PIC 9(4) COMP  VALUE ←
40. ADABAS
000176          03 ISN-BUFFER-LENGTHEMPL    PIC 9(4) COMP  VALUE ←

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

4.      ADABAS
000177      03 COMMAND-OPTION-1EMPL      PIC X      VALUE ←
SPACE.      ADABAS      IMP
000178      03 COMMAND-OPTION-2EMPL      PIC X      VALUE ←
SPACE.      ADABAS      IMP
000179      03 ADDITIONS-1EMPL      VALUE ←
SPACE.      ADABAS      IMP
000180      04 ADDITIONS-1-12EMPL PIC XX.      ←
      ADABAS
000181      04 FILLER PIC XX.      ←
      ADABAS
000182      04 ADDITIONS-1-58EMPL PIC X(4).      ←
      ADABAS
000183      03 FILLER REDEFINES ADDITIONS-1EMPL.      ←
      ADABAS      179
000184      04 ADDITIONS-1-BNEMPL PIC 9(4) COMP.      ←
      ADABAS
000185      04 FILLER PIC X(6).      ←
      ADABAS
000186      03 ADDITIONS-2EMPL      PIC X(4)      VALUE ←
SPACE.      ADABAS      IMP
000187      03 ADDITIONS-3EMPL      PIC X(8)      VALUE ←
SPACE.      ADABAS      IMP
000188      03 ADDITIONS-4EMPL      PIC X(8)      VALUE ←
SPACE.      ADABAS      IMP
000189      03 ADDITIONS-5EMPL      .      ←
      ADABAS
000190      04 ADDITIONS-5-BNEMPL PIC 9(9) COMP VALUE 0.      ←
      ADABAS
000191      04 ADDITIONS-5-58EMPL PIC X(4) VALUE SPACE.      ←
      ADABAS      IMP
000192      03 FILLER REDEFINES ADDITIONS-5EMPL.      ←
      ADABAS      189
000193      04 ADDITIONS-5-1EMPL PIC X.      ←
      ADABAS
000194      04 ADDITIONS-5-28EMPL PIC X(7).      ←
      ADABAS
000195      03 COMMAND-TIMEEMPL      PIC 9(9) COMP.      ←
      ADABAS
000196      03 USER-AREAEMPL      PIC X(4)      VALUE ' ←
'.      ADABAS
000197      01 ISNSIZEEMPL      PIC 9(9) COMP.      ←
      ADABAS
000198      01 ISNMOREEMPL      PIC 9(9) COMP.      ←
      ADABAS
000199      01 ISNINDEEMPL      PIC 9(4) COMP.      ←
      ADABAS
000200      01 EOF-COBEMPL      PIC 9 VALUE 0.      ←
      ADABAS
000201      88 EOFEMPL      VALUE 1.      ←
      ADABAS
000202      88 NOT-EOFEMPL      VALUE 0.      ←

```



```

ADABAS
000203      01  SAVE-DBID-1EMPL                PIC 9(9) COMP  VALUE  ↵
0. ADABAS
000204      01  SAVE-DBID-DEFEMPL REDEFINES SAVE-DBID-1EMPL.  ↵
ADABAS  203
000205      02  FILLER PIC X(2).  ↵
ADABAS
000206      02  SAVE-DBIDEMPL PIC 9(4) COMP.  ↵
ADABAS
000207      *
000208      *
000209      *          EXEC ADABAS
000210      *  DECLARE EMPL CURSOR FOR
000211      *  SELECT PERSONNEL-ID, NAME, FIRST-NAME, SEX
000212      *  FROM EMPLOYEES, VEHICLES
000213      *  WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
000214      *          AND PERSONNEL-ID BETWEEN '10000001' AND '19999999'
000215      *          AND VEHICLES.MODEL-YEAR-MAKE > ↵
:START-MODEL-YEAR-MAKE
000216      *          END-EXEC

```

```

000217      *
000218      *
00000280
000219      *          PROCEDURE DIVISION.  ↵
00000290
000220      *
000221      *
000222      *          EXEC ADABAS
000223      *  TRACE ON
000224      *          END-EXEC
000225      *
000226      *          DISPLAY HEADER.  ↵
00000300  19
000227      *          DISPLAY HEADER2.  ↵
00000310  27
000228      *          DISPLAY SPACE-LINE.  ↵
00000320  28
0 000229      *
000230      *
000231      *          EXEC ADABAS
000232      *  OPEN EMPL
000233      *          END-EXEC
000234      *
000235      *          MOVE '10000001' TO V-PERSONNEL-ID-F OF VALUE-BUFEMPL  ↵
ADABAS  146 145
000236      *          MOVE '19999999' TO V-PERSONNEL-ID-T OF VALUE-BUFEMPL  ↵
ADABAS  148 145
000237      *          MOVE START-MODEL-YEAR-MAKE TO V-MODEL-YEAR-MAKE OF  ↵
ADABAS  15 150
000238      *          VALUE-BUFEMPL  ↵

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS 145
000239 MOVE ADA-HALF-INTOPN TO FILLER1-CHAREMPL ↵
ADABAS 125 159
000240 COMPUTE ISNSIZEEMPL = ISN-BUFFER-LENGTHEMPL / 4 ↵
ADABAS 197 176
000241 MOVE 1 TO ISNINDEEMPL ↵
ADABAS 199
000242 MOVE 0 TO ISN-LOWER-LIMITEMPL ↵
ADABAS 168
000243 MOVE 0 TO ISN-QUANTITYEMPL ↵
ADABAS 169
000244 MOVE ' ' TO COMMAND-OPTION-2EMPL ↵
ADABAS 178
000245 MOVE ' ' TO COMMAND-OPTION-1EMPL ↵
ADABAS 177
000246 MOVE 0 TO ISN-BUFFER-LENGTHEMPL ↵
ADABAS 176
000247 MOVE SAVE-DBIDEMPL TO RESPONSE-CODEEMPL ↵
ADABAS 206 166
000248 MOVE 'S1' TO COMMAND-CODEEMPL ↵
ADABAS 160
000249 CALL 'ADABAS' USING ↵
ADABAS EXT
000250 CONTROL-BLOCKEMPL FORMAT-BUFEMPL RECORD-BUFEMPL ↵
ADABAS 157 130 137
000251 SEARCH-BUFEMPL VALUE-BUFEMPL ↵
ADABAS 133 145
000252 ISN-BUFEMPL ↵
ADABAS 155
000253 MOVE ISNEMPL TO ISN OF ↵
ADABAS 167 142
000254 EMPLOYEES ↵
ADABAS 136
000255 MOVE RESPONSE-CODEEMPL TO RESPONSE-CODE OF ↵
ADABAS 166 144
000256 EMPLOYEES ↵
ADABAS 136
000257 MOVE ISN-QUANTITYEMPL TO QUANTITY OF ↵
ADABAS 169 143
000258 EMPLOYEES ↵
ADABAS 136
000259 MOVE 0 TO ISNINDEEMPL ↵
ADABAS 199
000260 IF RESPONSE-CODEEMPL NOT = 0 ↵
ADABAS 166
000261 MOVE ' ' TO CSEQ ↵
ADABAS 112
000262 MOVE ' EXEC ADABAS ' TO ↵
CLN1V (01) ADABAS 114
000263 MOVE ' ' TO ↵
CLN2V (01) ADABAS 116
000264 MOVE ' OPEN EMPL ' TO ↵

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

CLN1V (02) ADABAS 114
  000265 1 MOVE ' ' TO ←
CLN2V (02) ADABAS 116
  000266 1 MOVE ' END-EXEC ' TO ←
CLN1V (03) ADABAS 114
  000267 1 MOVE ' ' TO ←
CLN2V (03) ADABAS 116
  000268 1 MOVE 03 TO CLNNUM ←
    ADABAS 117
  000269 1 CALL 'RESPINT' ←
    ADABAS EXT
000270 1 USING CONTROL-BLOCKEMPL DDFILE CSEQ FORMAT-BUFEMPL ←
    ADABAS 157 110 112 130
000271 1 RECORD-BUFEMPL SEARCH-BUFEMPL VALUE-BUFEMPL ←
    ADABAS 137 133 145
000272 1 CLN1 CLN2 TRCE CLNNUM DDBID. ←
    ADABAS 113 115 118 117 111
000273 MOVE ISN-QUANTITYEMPL TO ISNMOREEMPL ←
    ADABAS 169 198
000274 IF ISNMOREEMPL > 0 MOVE 0 TO EOF-COBEMPL ←
    ADABAS 198 200
000275 1 ELSE MOVE 1 TO EOF-COBEMPL. ←
    ADABAS 200
000276 IF ISNMOREEMPL < ISNSIZEEMPL ←
    ADABAS 198 197
000277 1 MOVE ISNMOREEMPL TO ISNSIZEEMPL. ←
    ADABAS 198 197
000278 *
000279 *
000280 * EXEC ADABAS
000281 * FETCH EMPL
000282 * END-EXEC
000283 *
000284 MOVE ADA-HALF-INTOPN TO FILLER1-CHAREMPL ←
    ADABAS 125 159
000285 IF ISNINDEMPL = ISNMOREEMPL MOVE 1 TO EOF-COBEMPL. ←
    ADABAS 199 198 200
0 000286 IF NOT-EOFEMPL ←
    ADABAS 202
000287 1 MOVE 0 TO EOF-COBEMPL ←
    ADABAS 200
000288 1 MOVE 'N' TO COMMAND-OPTION-2EMPL ←
    ADABAS 178
000289 1 MOVE ' ' TO COMMAND-OPTION-1EMPL ←
    ADABAS 177
000290 1 MOVE SAVE-DBIDEMPL TO RESPONSE-CODEEMPL ←
    ADABAS 206 166
000291 1 MOVE 'L1' TO COMMAND-CODEEMPL ←
    ADABAS 160
000292 1 CALL 'ADABAS' USING ←
    ADABAS EXT
000293 1 CONTROL-BLOCKEMPL FORMAT-BUFEMPL RECORD-BUFEMPL ←

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS 157 130 137
000294 1 SEARCH-BUFEMPL VALUE-BUFEMPL ↵
ADABAS 133 145
000295 1 ISN-BUFEMPL ↵
ADABAS 155
000296 1 MOVE ISNEMPL TO ISN OF ↵
ADABAS 167 142
000297 1 EMPLOYEES ↵
ADABAS 136
000298 1 MOVE RESPONSE-CODEEMPL TO RESPONSE-CODE OF ↵
ADABAS 166 144
000299 1 EMPLOYEES ↵
ADABAS 136
000300 1 IF RESPONSE-CODEEMPL = 3 ↵
ADABAS 166
000301 2 MOVE 1 TO EOF-COBEMPL ↵
ADABAS 200
000302 2 ELSE IF RESPONSE-CODEEMPL NOT = 0 ↵
ADABAS 166
000303 3 MOVE ' ' TO CSEQ ↵
ADABAS 112
000304 3 MOVE ' EXEC ADABAS ' TO ↵
CLN1V (01) ADABAS 114
000305 3 MOVE ' ' TO ↵
CLN2V (01) ADABAS 116
000306 3 MOVE ' FETCH EMPL ' TO ↵
CLN1V (02) ADABAS 114
000307 3 MOVE ' ' TO ↵
CLN2V (02) ADABAS 116
000308 3 MOVE ' END-EXEC ' TO ↵
CLN1V (03) ADABAS 114
000309 3 MOVE ' ' TO ↵
CLN2V (03) ADABAS 116
000310 3 MOVE 03 TO CLNNUM ↵
ADABAS 117
000311 3 CALL 'RESPINT' ↵
ADABAS EXT
000312 3 USING CONTROL-BLOCKEMPL DDFILE CSEQ FORMAT-BUFEMPL ↵
ADABAS 157 110 112 130
000313 3 RECORD-BUFEMPL SEARCH-BUFEMPL VALUE-BUFEMPL ↵
ADABAS 137 133 145
000314 3 CLN1 CLN2 TRCE CLNNUM DDBBID. ↵
ADABAS 113 115 118 117 111
000315 IF EOFEMPL MOVE 003 TO ADACODE ↵
ADABAS 201 44
000316 1 ELSE MOVE 0 TO ADACODE. ↵
ADABAS 44

```

```

000317          *
000318          PERFORM READ-EMPLOYEES UNTIL ADACODE = 3.
      00000330  383  44
000319          *
000320          *
000321          *      EXEC ADABAS
000322          *      CLOSE EMPL
000323          *      END-EXEC
000324          *
000325          MOVE ADA-HALF-INTOPN TO FILLER1-CHAREMPL
      ADABAS    125  159
000326          MOVE 'I' TO COMMAND-OPTION-1EMPL
      ADABAS    177
000327          MOVE 'S' TO COMMAND-OPTION-2EMPL
      ADABAS    178
000328          MOVE SAVE-DBIDEMPL TO RESPONSE-CODEEMPL
      ADABAS    206  166
000329          MOVE 'RC' TO COMMAND-CODEEMPL
      ADABAS    160
000330          CALL 'ADABAS' USING
      ADABAS    EXT
000331          CONTROL-BLOCKEMPL FORMAT-BUFEMPL RECORD-BUFEMPL
      ADABAS    157  130  137
000332          SEARCH-BUFEMPL VALUE-BUFEMPL
      ADABAS    133  145
000333          ISN-BUFEMPL
      ADABAS    155
000334          MOVE ISNEMPL TO ISN OF
      ADABAS    167  142
000335          EMPLOYEES
      ADABAS    136
000336          MOVE RESPONSE-CODEEMPL TO RESPONSE-CODE OF
      ADABAS    166  144
000337          EMPLOYEES
      ADABAS    136
000338          IF RESPONSE-CODEEMPL NOT = 0
      ADABAS    166
000339          1      MOVE '      ' TO CSEQ
      ADABAS    112
000340          1      MOVE '      EXEC ADABAS      ' TO
CLN1V (01) ADABAS  114
000341          1      MOVE '      ' TO
CLN2V (01) ADABAS  116
000342          1      MOVE '      CLOSE EMPL      ' TO
CLN1V (02) ADABAS  114
0 000343          1      MOVE '      ' TO
CLN2V (02) ADABAS  116
000344          1      MOVE '      END-EXEC      ' TO
CLN1V (03) ADABAS  114
000345          1      MOVE '      ' TO
CLN2V (03) ADABAS  116

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

000346      1          MOVE 03 TO CLNNUM                                ↵
      ADABAS  117
000347      1          CALL 'RESPINT'                                  ↵
      ADABAS  EXT
000348      1          USING CONTROL-BLOCKEMPL DDFILE CSEQ FORMAT-BUFEMPL ↵
      ADABAS  157 110 112 130
000349      1          RECORD-BUFEMPL SEARCH-BUFEMPL VALUE-BUFEMPL ↵
      ADABAS  137 133 145
000350      1          CLN1 CLN2 TRCE CLNNUM DDBBID.                  ↵
      ADABAS  113 115 118 117 111
000351          *
000352          *
000353          *          EXEC ADABAS
000354          *          DBCLOSE
000355          *          END-EXEC
000356          *
000357          MOVE ADA-HALF-INTOPN TO FILLER1-CHAROPN                ↵
      ADABAS  125 47
000358          MOVE 1500 TO RECORD-BUFFER-LENGTHOPN                 ↵
      ADABAS  60
000359          MOVE ' ' TO COMMAND-OPTION-2OPN                       ↵
      ADABAS  66
000360          MOVE ' ' TO COMMAND-OPTION-1OPN                       ↵
      ADABAS  65
000361          MOVE SAVE-DBIDOPN TO RESPONSE-CODEOPN                 ↵
      ADABAS  129 54
000362          MOVE 'CL' TO COMMAND-CODEOPN                          ↵
      ADABAS  48
000363          CALL 'ADABAS' USING                                    ↵
      ADABAS  EXT
000364          CONTROL-BLOCKOPN FORMAT-BUFOPN RECORD-BUFOPN ↵
      ADABAS  45 89 94
000365          SEARCH-BUFOPN VALUE-BUFOPN                            ↵
      ADABAS  90 91
000366          ISN-BUFOPN                                            ↵
      ADABAS  92
000367          IF RESPONSE-CODEOPN NOT = 0                            ↵
      ADABAS  54
000368      1          MOVE ' ' TO CSEQ                                ↵
      ADABAS  112
000369      1          MOVE '          EXEC ADABAS          ' TO ↵
CLN1V (01) ADABAS  114
000370      1          MOVE '          ' TO ↵
CLN2V (01) ADABAS  116
000371      1          MOVE '          DBCLOSE          ' TO ↵
CLN1V (02) ADABAS  114
000372      1          MOVE '          ' TO ↵
CLN2V (02) ADABAS  116
000373      1          MOVE '          END-EXEC          ' TO ↵
CLN1V (03) ADABAS  114
000374      1          MOVE '          ' TO ↵
CLN2V (03) ADABAS  116

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

000375      1      MOVE 03 TO CLNNUM      ↵
      ADABAS  117
000376      1      CALL 'RESPINT'      ↵
      ADABAS  EXT
000377      1      USING CONTROL-BLOCKOPN  DDFILE CSEQ FORMAT-BUFOPN  ↵
      ADABAS  45 110 112 89
000378      1      RECORD-BUFOPN  SEARCH-BUFOPN  VALUE-BUFOPN  ↵
      ADABAS  94 90 91
000379      1      CLN1 CLN2 TRCE CLNNUM DDBBID.      ↵
      ADABAS  113 115 118 117 111

```

```

000380      *
000381      STOP RUN.      ↵
      00000340
000382      *
000383      READ-EMPLOYEES.      ↵
      00000350
000384      MOVE PERSONNEL-ID TO PERSONNEL-NR.      ↵
      00000360 138 31
000385      MOVE NAME TO LAST-NAME.      ↵
      00000370 139 33
000386      MOVE FIRST-NAME TO F-NAME.      ↵
      00000380 140 35
000387      MOVE SEX TO KIND.      ↵
      00000390 141 37
000388      DISPLAY LINE1.      ↵
      00000400 29
000389      MOVE SPACE TO LINE1.      ↵
      00000410 IMP 29
000390      *
000391      *      ↵
      00000420
000392      *      EXEC ADABAS      ↵
      00000430
000393      *      FETCH EMPL      ↵
      00000440
000394      *      END-EXEC      ↵
      00000450
000395      *
      00000460
000396      MOVE ADA-HALF-INTOPN TO FILLER1-CHAREMPL      ↵
      ADABAS  125 159
000397      IF ISNINDEMPL = ISNMOREEMPL MOVE 1 TO EOF-COBEMPL.      ↵
      ADABAS  199 198 200
000398      IF NOT-EOFEMPL      ↵
      ADABAS  202
000399      MOVE 0 TO EOF-COBEMPL      ↵
      ADABAS  200
0 000400      1      MOVE 'N' TO COMMAND-OPTION-2EMPL      ↵
      ADABAS  178
000401      1      MOVE ' ' TO COMMAND-OPTION-1EMPL      ↵

```

APPENDIX G - EXAMPLE OF COBOL CODE GENERATED BY ADABAS NATIVE SQL

```

ADABAS 177
000402 1 MOVE SAVE-DBIDEMPL TO RESPONSE-CODEEMPL ↵
ADABAS 206 166
000403 1 MOVE 'L1' TO COMMAND-CODEEMPL ↵
ADABAS 160
000404 1 CALL 'ADABAS' USING ↵
ADABAS EXT
000405 1 CONTROL-BLOCKEMPL FORMAT-BUFEMPL RECORD-BUFEMPL ↵
ADABAS 157 130 137
000406 1 SEARCH-BUFEMPL VALUE-BUFEMPL ↵
ADABAS 133 145
000407 1 ISN-BUFEMPL ↵
ADABAS 155
000408 1 MOVE ISNEMPL TO ISN OF ↵
ADABAS 167 142
000409 1 EMPLOYEES ↵
ADABAS 136
000410 1 MOVE RESPONSE-CODEEMPL TO RESPONSE-CODE OF ↵
ADABAS 166 144
000411 1 EMPLOYEES ↵
ADABAS 136
000412 1 IF RESPONSE-CODEEMPL = 3 ↵
ADABAS 166
000413 2 MOVE 1 TO EOF-COBEMPL ↵
ADABAS 200
000414 2 ELSE IF RESPONSE-CODEEMPL NOT = 0 ↵
ADABAS 166
000415 3 MOVE ' ' TO CSEQ ↵
ADABAS 112
000416 3 MOVE ' EXEC ADABAS ' TO ↵
CLN1V (01) ADABAS 114
000417 3 MOVE ' ' TO ↵
CLN2V (01) ADABAS 116
000418 3 MOVE ' FETCH EMPL ' TO ↵
CLN1V (02) ADABAS 114
000419 3 MOVE ' ' TO ↵
CLN2V (02) ADABAS 116
000420 3 MOVE ' END-EXEC ' TO ↵
CLN1V (03) ADABAS 114
000421 3 MOVE ' ' TO ↵
CLN2V (03) ADABAS 116
000422 3 MOVE 03 TO CLNNUM ↵
ADABAS 117
000423 3 CALL 'RESPINT' ↵
ADABAS EXT
000424 3 USING CONTROL-BLOCKEMPL DDFILE CSEQ FORMAT-BUFEMPL ↵
ADABAS 157 110 112 130
000425 3 RECORD-BUFEMPL SEARCH-BUFEMPL VALUE-BUFEMPL ↵
ADABAS 137 133 145
000426 3 CLN1 CLN2 TRCE CLNNUM DDBID. ↵
ADABAS 113 115 118 117 111
000427 IF EOFEMPL MOVE 003 TO ADACODE ↵

```



```
ADABAS 201 44  
000428 1 ELSE MOVE 0 TO ADACODE.  
ADABAS 44
```


15

APPENDIX H - FORTRAN EXAMPLES

▪ Example 1	316
▪ Example 2	317
▪ Example 3	320

This chapter covers the following topics:

Example 1

```

PROGRAM FEX1
C   AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH
C   CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS PERSONNEL-ID
C   NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,
C   PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE
C   FOLLOWING CONDITION:
C       PERSONNEL-ID BETWEEN 10000001 AND 19999999
C       MODEL-YEAR-MAKE >
C       CLASS = 'C'
CHARACTER*22  STARTS
CHARACTER*20  STARTM  /'MERCEDES BENZ'/
CHARACTER*2   STAYM   /'86'/
EQUIVALENCE (STARTS,STARTM)
EQUIVALENCE (STARTS(21:21),STAYM)
C
C       EXEC ADABAS
BEGIN DECLARE SECTION
C       END-EXEC
C
C       EXEC ADABAS
DECLARE EMPL CURSOR FOR
SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX
FROM EMPLOYEES, VEHICLES
WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
AND PERSONNEL-ID BETWEEN '10000001' AND '19999999'
AND VEHICLES.MODEL-YEAR-MAKE > :STARTS
AND VEHICLES.CLASS = 'C'
C       END-EXEC
C
C       WRITE (6,10)
C
C       EXEC ADABAS
OPEN EMPL
C       END-EXEC
C
C       EXEC ADABAS
FETCH EMPL
C       END-EXEC
C
1 IF (SQLCOD .EQ. 3) GOTO 2
C
C       WRITE (6,20) PID,NAME,FNAME,BIRTH,SEX
C
C       EXEC ADABAS

```

```

        FETCH EMPL
        END-EXEC
C
        GOTO 1
C

        2 CONTINUE
C
        EXEC ADABAS
        CLOSE EMPL
        END-EXEC
C
        EXEC ADABAS
        DBCLOSE
        END-EXEC
C
        10 FORMAT ('1PERSONNEL-ID',8X,'NAME',13X,'FIRST-NAME',8X,
          *      'BIRTH',1X,'SEX' / 1X,64('*') / )
        20 FORMAT (3X,A8,3X,A20,1X,A20,1X,A6,1X,A1)
C
        END

```

Example 2

```

        PROGRAM FEX2
C
        DELETE AN EMPLOYEE RECORD AND RELEASE ALL CARS WHICH ARE
C
        ASSIGNED TO THIS EMPLOYEE. A PRIVATE CARS WILL BE DELETED
C
        AND A COMPANY CAR WILL BE MADE A POOL-CAR WHICH IS IDENTIFIED
C
        BY ITS PERSONNEL-ID CONTAINING ONLY THE COUNTRY CODE.
C
        CHARACTER*8  PERSNR  /'20007100'/
        INTEGER*4    EMPISN
        CHARACTER*15 CNUM
        CHARACTER*1  CNO
        EQUIVALENCE (CNUM,CNO)
C
        EXEC ADABAS
        BEGIN DECLARE SECTION
        END-EXEC
C
        EXEC ADABAS
        READ LOGICAL
        DECLARE VEH1 CURSOR FOR
        SELECT REG-NUM, PERSONNEL-ID, CLASS
        FROM VEHICLES
        WHERE PERSONNEL-ID GE :PERSNR
        OPTIONS HOLD

```

```
ORDER BY PERSONNEL-ID
      END-EXEC
C
C   FIND EMPLOYEE
C
      EXEC ADABAS
      FIND
      SELECT
      FROM EMPLOYEES EMPL1
      WHERE PERSONNEL-ID = :PERSNR
      OPTIONS HOLD
      END-EXEC
C
C   IF THE PERSONNEL-ID EXISTS DELETE THE EMPLOYEE AND READ THE
C   VEHICLES FILE
C
      IF (SQLQTY .EQ. 1) THEN
          EMPISN = SQLISN
          GOTO 3
1       GOTO 4
      ELSE
          WRITE (6,10) PERSNR
      END IF
C
2     CONTINUE
C
      EXEC ADABAS
      DBCLOSE
      END-EXEC
C
      STOP
```

```
C
C*** DELETE EMPLOYEE
C
      3 CONTINUE
C
      EXEC ADABAS
      DELETE
      FROM EMPLOYEES
      WHERE ISN = :EMPISN
      END-EXEC
C
      WRITE (6,20) PERSNR
C
      GOTO 1
C
C*** DEALLOCATE CARS
C
      4 CONTINUE
```

```
C      EXEC ADABAS
OPEN VEH1
      END-EXEC
C
      EXEC ADABAS
FETCH VEH1
      END-EXEC
C
5 IF (SQLCOD .EQ. 3 .OR. PID .NE. PERSNR) GOTO 6
C
      IF (CLASS .EQ. 'P') THEN
          EXEC ADABAS
          DELETE
          FROM VEHICLES
          WHERE CURRENT OF VEH1
          END-EXEC
          WRITE (6,30) REGNUM
      ELSE
          CNUM = PID
          PID = CNO
          EXEC ADABAS
          UPDATE VEHICLES
          WHERE CURRENT OF VEH1
          END-EXEC
          WRITE (6,40) REGNUM
      END IF
C
      EXEC ADABAS
FETCH VEH1
      END-EXEC
C
      GOTO 5
C
6 CONTINUE
C
      EXEC ADABAS
CLOSE VEH1
      END-EXEC
C
      EXEC ADABAS
COMMIT WORK
      END-EXEC
C
      GOTO 2
```

```

C
10 FORMAT (' NO EMPLOYEE FOUND WITH PERSONNEL-ID ',A8)
20 FORMAT (' EMPLOYEE ',A8,' HAS BEEN DELETED')
30 FORMAT (' PRIVATE CAR ',A15,' HAS BEEN DELETED')
40 FORMAT (' COMPANY CAR ',A15,' HAS BEEN UPDATED')
END

```

Example 3

```

PROGRAM FEX3
C   SALARY INCREASE.
C   THIS PROGRAM INCREASES THE SALARY OF EVERY EMPLOYEE BY
C   4 PERCENT.
C   THE DEPARTMENT, THE OVERALL AMOUNT OF PAY RISE FOR THE
C   DEPARTMENT AND THE PAY RISE FOR ALL DEPARTMENTS WILL BE PRINTED
C   OUT.
C   THE PROGRAM IS RESTARTABLE. AFTER AN ABNORMAL TERMINATION THE
C   PROGRAM EXECUTION WOULD RESTART WITH THE LAST DEPARTMENT
C   WHOSE SALARY UPDATE HAD BEEN COMPLETED BEFORE THE ABEND
C   OCCURED.
C
C   CHARACTER*10 COMDAT
C   CHARACTER*6  COMDEP
C   INTEGER*4   COMSUM
C   EQUIVALENCE (COMDAT,COMDEP)
C   EQUIVALENCE (COMDAT(7:7),COMSUM)
C   CHARACTER*6  SDEP
C   INTEGER*4   IND, I, J, NEWSAL, INCRS, SUMDEP, SUMTOT, E1QTY
C
C   EXEC ADABAS
BEGIN DECLARE SECTION
C   END-EXEC
C
C   EXEC ADABAS
HISTOGRAM
DECLARE EMP1 CURSOR FOR
SELECT  DEPT
FROM EMPLOYEES E1
WHERE DEPT GE :COMDEP
OPTIONS PREFIX=E1
GROUP BY DEPT
C   END-EXEC
C
C   EXEC ADABAS
READ LOGICAL
DECLARE EMP2 CURSOR FOR
SELECT PERSONNEL-ID, DEPT, SALARY, INCOME(COUNT)

```



```

FROM EMPLOYEES
WHERE DEPT GE :SDEP
OPTIONS HOLD
ORDER BY DEPT
      END-EXEC
C
      EXEC ADABAS
CONNECT 'INCREASE'
UPD=EMPLOYEES
AND USERDATA INTO :COMDAT
      END-EXEC
C
C A HISTOGRAM STATEMENT IS USED TO ASCERTAIN THE NUMBER OF
C EMPLOYEES PER DEPARTMENT
C
      EXEC ADABAS
OPEN EMP1
      END-EXEC
C
      EXEC ADABAS
FETCH EMP1
      END-EXEC
E1QTY = SQLQTY
C
IF (COMDAT .NE. ' ') THEN
C
C RESTART PROCESSING
C
WRITE (6,*) 'LAST PROGRAM RUN TERMINATED ABNORMALLY'
WRITE (6,50) COMDEP
C
      EXEC ADABAS
FETCH EMP1
      END-EXEC.
E1QTY = SQLQTY
END IF
C
SDEP = E1DEPT
C
      EXEC ADABAS
OPEN EMP2
      END-EXEC
C
WRITE (6,10)
C
1 IF (SQLCOD .EQ. 3) GOTO 4
C
C THE EMPLOYEES FILE WILL BE READ UNTIL ALL RECORDS FOR THE
C DEPARTMENT HAVE BEEN PROCESSED AND THE SALARY HAS BEEN
C UPDATED

```

```
C
  DO 3 J=1, E1QTY
    EXEC ADABAS
    FETCH EMP2
    END-EXEC
C
    THE SALARY INCREASE CAN BE EXECUTED WHEN THE COUNT OF THE
C
    PERIODIC GROUP IS LESS THAN 40.
    IF (CINC .LT. 40) THEN
      INCRS = NINT(REAL(SALARY(1)) * 0.04)
      NEWSAL = SALARY(1) + INCRS
      IND = CINC + 1
C
      DO 2 I = CINC, 0, -1
        SALARY(IND) = SALARY(I)
        IND = IND - 1
2
      CONTINUE
C
      SALARY(1) = NEWSAL
C
      EXEC ADABAS
      UPDATE EMPLOYEES
      WHERE CURRENT OF EMP2
      END-EXEC
C
      SUMDEP = SUMDEP + INCRS
      SUMTOT = SUMTOT + INCRS
    ELSE
      WRITE (6,40) PID
    END IF
C
3 CONTINUE
```

```
C
  WRITE (6,20) DEPT, SUMDEP
  SUMDEP = 0
C
  COMDEP = DEPT
  COMSUM = SUMTOT
  EXEC ADABAS
  COMMIT WORK
  USERDATA = :COMDAT
  END-EXEC
C
  EXEC ADABAS
  FETCH EMP1
  END-EXEC
  E1QTY = SQLQTY
C
  GOTO 1
C
4 CONTINUE
```

```
C      EXEC ADABAS
      CLOSE EMP1
      END-EXEC

C      EXEC ADABAS
      CLOSE EMP2
      END-EXEC

C      WRITE (6,30) SUMTOT
      COMDAT = ' '

C      EXEC ADABAS
      DBCLOSE
      USERDATA = :COMDAT
      END-EXEC

C      10 FORMAT (' DEPARTMENT',15X,'SALARY INCREASE'/1X,40('*'))
      20 FORMAT (4X,A6,16X,I10)
      30 FORMAT (/50('-')// ' TOTAL SALARY INCREASE : ',I11)
      40 FORMAT (' UPDATE PERSON ',A8,' NOT POSSIBLE')
      50 FORMAT (' LAST DEPARTMENT WAS ',A6)
      END
```


16 APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

```

PROGRAM FEX1                                00000010
C      AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH 00000020
C      CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS PERSONNEL-ID 00000030
C      NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE, 00000040
C      PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE 00000050
C      FOLLOWING CONDITION: 00000060
C      PERSONNEL-ID BETWEEN 10000001 AND 19999999 00000070
C      MODEL-YEAR-MAKE > 00000080
C      CLASS = 'C' 00000090
CHARACTER*22 STARTS 00000100
CHARACTER*20 STARTM /'MERCEDES BENZ'/ 00000110
CHARACTER*2 STAYM /'86'/ 00000120
EQUIVALENCE (STARTS,STARTM) 00000130
EQUIVALENCE (STARTS(21:21),STAYM) 00000140
C 00000150
* 00000160
* EXEC ADABAS 00000170
* BEGIN DECLARE SECTION 00000180
* END-EXEC 00000190
* 00000200
INTEGER*2 SQLCOD /0/ ADABAS
CHARACTER* 2 SQC000 /'AS' / ADABAS
CHARACTER* 2 SQCC00 /' ' / ADABAS
CHARACTER* 4 SQCI00 /'OPEN' / ADABAS
INTEGER* 2 SQCF00 / 0 / ADABAS
INTEGER* 2 SQCR00 / 0 / ADABAS
INTEGER* 4 SQCS00 / 0 / ADABAS
INTEGER* 4 SQCL00 / 0 / ADABAS
INTEGER* 4 SQCQ00 / 0 / ADABAS
INTEGER* 2 SQC300 / 0 / ADABAS
INTEGER* 2 SQC400 / 0 / ADABAS
INTEGER* 2 SQC500 / 0 / ADABAS

```

APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

INTEGER*	2	SQC600	/	0	/	ADABAS
INTEGER*	2	SQC700	/	4	/	ADABAS
CHARACTER*	1	SQC100	/'	'	/	ADABAS
CHARACTER*	1	SQC200	/'	'	/	ADABAS
CHARACTER*	8	SQCM00	/'	'	/	ADABAS
CHARACTER*	4	SQCN00	/'	'	/	ADABAS
CHARACTER*	8	SQC000	/'	'	/	ADABAS
INTEGER*	2	SQCE00	/	0	/	ADABAS
INTEGER*	2	SQCH00	/	0	/	ADABAS
INTEGER*	2	SQCJ00	/	0	/	ADABAS
INTEGER*	2	SQCK00	/	0	/	ADABAS
INTEGER*	4	SQC800	/	0	/	ADABAS
INTEGER*	4	SQCT00	/	0	/	ADABAS
CHARACTER*	4	SQCU00	/'	AS	'	ADABAS
CHARACTER*	4	SQC900	/'	'	/	ADABAS
CHARACTER*	1	SQCD00				ADABAS
CHARACTER*	2	SQCV00				ADABAS
CHARACTER*	4	SQCW00				ADABAS
CHARACTER*	8	SQCP00				ADABAS
CHARACTER*	7	SQCG00				ADABAS
CHARACTER*	1	SQCZ00				ADABAS
CHARACTER*	80	SQCB00				ADABAS
INTEGER*	2	SQCY00				ADABAS
CHARACTER*	1	SQCA00(00080)				ADABAS
EQUIVALENCE		(SQCA00(00001),SQCB00)				ADABAS
EQUIVALENCE		(SQCA00(00001),SQC000)				ADABAS
EQUIVALENCE		(SQCA00(00003),SQCC00)				ADABAS
EQUIVALENCE		(SQCA00(00005),SQCI00)				ADABAS
EQUIVALENCE		(SQCA00(00009),SQCF00)				ADABAS
EQUIVALENCE		(SQCA00(00009),SQCD00)				ADABAS
EQUIVALENCE		(SQCA00(00011),SQCR00)				ADABAS
EQUIVALENCE		(SQCA00(00013),SQCS00)				ADABAS
EQUIVALENCE		(SQCA00(00017),SQCL00)				ADABAS
EQUIVALENCE		(SQCA00(00021),SQCQ00)				ADABAS
EQUIVALENCE		(SQCA00(00025),SQC300)				ADABAS
EQUIVALENCE		(SQCA00(00027),SQC400)				ADABAS
EQUIVALENCE		(SQCA00(00029),SQC500)				ADABAS
EQUIVALENCE		(SQCA00(00031),SQC600)				ADABAS
EQUIVALENCE		(SQCA00(00033),SQC700)				ADABAS
EQUIVALENCE		(SQCA00(00035),SQC100)				ADABAS
EQUIVALENCE		(SQCA00(00036),SQC200)				ADABAS
EQUIVALENCE		(SQCA00(00037),SQCM00)				ADABAS
EQUIVALENCE		(SQCA00(00037),SQCV00)				ADABAS
EQUIVALENCE		(SQCA00(00037),SQCY00)				ADABAS
EQUIVALENCE		(SQCA00(00041),SQCW00)				ADABAS
EQUIVALENCE		(SQCA00(00045),SQCN00)				ADABAS
EQUIVALENCE		(SQCA00(00049),SQC000)				ADABAS
EQUIVALENCE		(SQCA00(00057),SQCP00)				ADABAS
EQUIVALENCE		(SQCA00(00057),SQCE00)				ADABAS
EQUIVALENCE		(SQCA00(00059),SQCH00)				ADABAS
EQUIVALENCE		(SQCA00(00061),SQCJ00)				ADABAS
EQUIVALENCE		(SQCA00(00063),SQCK00)				ADABAS

EQUIVALENCE	(SQCA00(00065),SQC800)	ADABAS
EQUIVALENCE	(SQCA00(00065),SQCZ00)	ADABAS
EQUIVALENCE	(SQCA00(00066),SQCG00)	ADABAS
EQUIVALENCE	(SQCA00(00069),SQC900)	ADABAS
EQUIVALENCE	(SQCA00(00073),SQCT00)	ADABAS
EQUIVALENCE	(SQCA00(00077),SQCU00)	ADABAS
CHARACTER*	1 SQFB00	ADABAS
CHARACTER*	1 SQSB00	ADABAS
CHARACTER*	1 SQVB00	ADABAS
CHARACTER*	1 SQDS00	ADABAS
CHARACTER*	500 SQRB00	ADABAS
CHARACTER*	10 SQDK00	ADABAS
CHARACTER*	3 SQDD00	/'030' / ADABAS
CHARACTER*	8 SQDA00	ADABAS
CHARACTER*	40 SQDB00(00020)	ADABAS
CHARACTER*	40 SQDC00(00020)	ADABAS
CHARACTER*	7 SQDE00	ADABAS
INTEGER*	2 SQDG00	ADABAS
INTEGER*	2 SQLRSP	ADABAS
INTEGER*	4 SQLQTY	ADABAS
INTEGER*	4 SQLISN	ADABAS
CHARACTER*	8 SQDLO0	ADABAS
INTEGER*	2 SQDN00	ADABAS
INTEGER*	4 SQD000	ADABAS
REAL*	4 SQDP00	ADABAS
REAL*	8 SQDQ00	ADABAS
CHARACTER*	1 SQDM00(00008)	ADABAS
EQUIVALENCE	(SQDM00(00001),SQDLO0)	ADABAS
EQUIVALENCE	(SQDM00(00001),SQDN00)	ADABAS
EQUIVALENCE	(SQDM00(00001),SQD000)	ADABAS
EQUIVALENCE	(SQDM00(00001),SQDP00)	ADABAS
EQUIVALENCE	(SQDM00(00001),SQDQ00)	ADABAS
INTEGER*	2 SQDR00	ADABAS
CHARACTER*	1 SQDF00	ADABAS
CHARACTER*	1 SQDT00(00002)	ADABAS
EQUIVALENCE	(SQDT00(00001),SQDR00)	ADABAS
EQUIVALENCE	(SQDT00(00002),SQDF00)	ADABAS
CHARACTER*	37 SQFB01	ADABAS
CHARACTER*	1 SQFA01(00037)	ADABAS
EQUIVALENCE	(SQFA01(00001),SQFB01)	ADABAS
CHARACTER*37	SQFC01/'AA,8,A,AE,20,A,AC,20,A,AH,6,U,AG,1,A.' /	ADABAS
EQUIVALENCE	(SQFA01(00001),SQFC01)	ADABAS
CHARACTER*	59 SQSB01	ADABAS
CHARACTER*	1 SQSA01(00059)	ADABAS
EQUIVALENCE	(SQSA01(00001),SQSB01)	ADABAS
CHARACTER*40	SQSC01/'(22,AA,24,AC)/22/AA,8,A,S,AA,8,A,D,/24/A' /	ADABAS
EQUIVALENCE	(SQSA01(00001),SQSC01)	ADABAS
CHARACTER*19	SQSD01/'0,22,A,GT,D,AH,1,A.' /	ADABAS
EQUIVALENCE	(SQSA01(00041),SQSD01)	ADABAS
EQUIVALENCE	(SQRA01(00001),SQRB01)	ADADATA
EQUIVALENCE	(SQRA01(00001),EMPLOY)	ADADATA
CHARACTER*	8 PID	ADADATA

APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

EQUIVALENCE	(SQRA01(00001),PID)		ADADATA
CHARACTER*	20 NAME		ADADATA
EQUIVALENCE	(SQRA01(00009),NAME)		ADADATA
CHARACTER*	20 FIRSTN		ADADATA
EQUIVALENCE	(SQRA01(00029),FIRSTN)		ADADATA
CHARACTER*	6 BIRTH		ADADATA
EQUIVALENCE	(SQRA01(00049),BIRTH)		ADADATA
CHARACTER*	1 SEX		ADADATA
EQUIVALENCE	(SQRA01(00055),SEX)		ADADATA
CHARACTER*	55 EMPLOY		ADADATA
CHARACTER*	55 SQRB01		ADADATA
CHARACTER*	1 SQRA01(00055)		ADADATA
EQUIVALENCE	(SQVA01(00001),SQVB01)		ADABAS
CHARACTER*	8 SQVC01	/' ' /	ADABAS
EQUIVALENCE	(SQVA01(00001),SQVC01)		ADABAS
CHARACTER*	8 SQVD01	/' ' /	ADABAS
EQUIVALENCE	(SQVA01(00009),SQVD01)		ADABAS
CHARACTER*	22 SQVE01	/' ' /	ADABAS
EQUIVALENCE	(SQVA01(00017),SQVE01)		ADABAS
CHARACTER*	1 SQVF01	/' ' /	ADABAS
EQUIVALENCE	(SQVA01(00039),SQVF01)		ADABAS
CHARACTER*	39 SQVB01		ADABAS
CHARACTER*	1 SQVA01(00039)		ADABAS
INTEGER*	4 SQDS01(00001)		ADABAS
CHARACTER*	2 SQC001	/'AS' /	ADABAS
CHARACTER*	2 SQCC01	/' ' /	ADABAS
CHARACTER*	4 SQCI01	/'EMPL' /	ADABAS
INTEGER*	2 SQCF01	/ 22 /	ADABAS
INTEGER*	2 SQCR01	/ 0 /	ADABAS
INTEGER*	4 SQCS01	/ 0 /	ADABAS
INTEGER*	4 SQCL01	/ 0 /	ADABAS
INTEGER*	4 SQCQ01	/ 0 /	ADABAS
INTEGER*	2 SQC301	/ 37 /	ADABAS
INTEGER*	2 SQC401	/ 55 /	ADABAS
INTEGER*	2 SQC501	/ 59 /	ADABAS
INTEGER*	2 SQC601	/ 39 /	ADABAS
INTEGER*	2 SQC701	/ 4 /	ADABAS
CHARACTER*	1 SQC101	/' ' /	ADABAS
CHARACTER*	1 SQC201	/' ' /	ADABAS
CHARACTER*	8 SQCM01	/' ' /	ADABAS
CHARACTER*	4 SQCN01	/' ' /	ADABAS
CHARACTER*	8 SQC001	/' ' /	ADABAS
CHARACTER*	8 SQCP01	/' ' /	ADABAS
INTEGER*	4 SQC801	/ 0 /	ADABAS
INTEGER*	4 SQCT01	/ 0 /	ADABAS
CHARACTER*	4 SQCU01	/'AS' /	ADABAS
CHARACTER*	4 SQC901	/' ' /	ADABAS
CHARACTER*	1 SQCD01		ADABAS
CHARACTER*	2 SQCV01		ADABAS
CHARACTER*	4 SQCW01		ADABAS
CHARACTER*	7 SQCG01		ADABAS
CHARACTER*	1 SQCZ01		ADABAS


```

CHARACTER*      80 SQCB01                                ADABAS
INTEGER*        2 SQCY01                                ADABAS
CHARACTER*      1 SQCA01(00080)                          ADABAS
EQUIVALENCE     (SQCA01(00001),SQCB01)                 ADABAS
EQUIVALENCE     (SQCA01(00001),SQC001)                 ADABAS
EQUIVALENCE     (SQCA01(00003),SQCC01)                 ADABAS
EQUIVALENCE     (SQCA01(00005),SQCI01)                 ADABAS
EQUIVALENCE     (SQCA01(00009),SQCF01)                 ADABAS
EQUIVALENCE     (SQCA01(00009),SQCD01)                 ADABAS
EQUIVALENCE     (SQCA01(00011),SQCR01)                 ADABAS
EQUIVALENCE     (SQCA01(00013),SQCS01)                 ADABAS
EQUIVALENCE     (SQCA01(00017),SQCL01)                 ADABAS
EQUIVALENCE     (SQCA01(00021),SQCQ01)                 ADABAS
EQUIVALENCE     (SQCA01(00025),SQC301)                 ADABAS
EQUIVALENCE     (SQCA01(00027),SQC401)                 ADABAS
EQUIVALENCE     (SQCA01(00029),SQC501)                 ADABAS
EQUIVALENCE     (SQCA01(00031),SQC601)                 ADABAS
EQUIVALENCE     (SQCA01(00033),SQC701)                 ADABAS
EQUIVALENCE     (SQCA01(00035),SQC101)                 ADABAS
EQUIVALENCE     (SQCA01(00036),SQC201)                 ADABAS
EQUIVALENCE     (SQCA01(00037),SQCM01)                 ADABAS
EQUIVALENCE     (SQCA01(00037),SQCV01)                 ADABAS
EQUIVALENCE     (SQCA01(00037),SQCY01)                 ADABAS
EQUIVALENCE     (SQCA01(00041),SQCW01)                 ADABAS
EQUIVALENCE     (SQCA01(00045),SQCN01)                 ADABAS
EQUIVALENCE     (SQCA01(00049),SQC001)                 ADABAS
EQUIVALENCE     (SQCA01(00057),SQCP01)                 ADABAS
EQUIVALENCE     (SQCA01(00065),SQC801)                 ADABAS
EQUIVALENCE     (SQCA01(00065),SQCZ01)                 ADABAS
EQUIVALENCE     (SQCA01(00066),SQCG01)                 ADABAS
EQUIVALENCE     (SQCA01(00069),SQC901)                 ADABAS
EQUIVALENCE     (SQCA01(00073),SQCT01)                 ADABAS
EQUIVALENCE     (SQCA01(00077),SQCU01)                 ADABAS
INTEGER*        4 SQDH01                                ADABAS
INTEGER*        4 SQDJ01                                ADABAS
INTEGER*        2 SQDI01                                ADABAS
LOGICAL*        1 SQEF01                                ADABAS
INTEGERS*       2 SQDR01                                ADABAS
CHARACTER*      1 SQDF01                                ADABAS
CHARACTER*      1 SQDT01(00002)                          ADABAS
EQUIVALENCE     (SQDT01(00001),SQDR01)                 ADABAS
EQUIVALENCE     (SQDT01(00002),SQDF01)                 ADABAS
C                                                        00000210
*                                                        00000220
*      EXEC ADABAS                                       00000230
*      DECLARE EMPL CURSOR FOR                           00000240
*      SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX 00000250
*      FROM EMPLOYEES, VEHICLES                          00000260
*      WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID 00000270
*           AND PERSONNEL-ID BETWEEN '10000001' AND '19999999' 00000280
*           AND VEHICLES.MODEL-YEAR-MAKE > :STARTS          00000290
*           AND VEHICLES.CLASS = 'C'                       00000300

```

APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

```

*          END-EXEC                                00000310
*                                                  00000320
C                                                  00000330
  WRITE (6,10)                                    00000340
C                                                  00000350
*                                                  00000360
*          EXEC ADABAS                             00000370
*          OPEN EMPL                               00000380
*          END-EXEC                                00000390
*                                                  00000400
          SQVC01 ='10000001'                       ADABAS
          SQVD01 ='19999999'                       ADABAS
          SQVE01 =STARTS                           ADABAS
          SQVF01 ='C'                               ADABAS
SQDA00='00000400'                                  ADABAS
SQDB00(01)='          EXEC ADABAS                '   ADABAS
SQDC00(01)='          '                           '   ADABAS
SQDB00(02)='          OPEN EMPL                   '   ADABAS
SQDC00(02)='          '                           '   ADABAS
SQDB00(03)='          END-EXEC                     '   ADABAS
SQDC00(03)='          '                           '   ADABAS
SQDG00=03                                          ADABAS
SQDH01=SQC701 / 4                                 ADABAS
SQDI01=1                                           ADABAS
SQCL01=0                                           ADABAS
SQCQ01=0                                           ADABAS
SQC201=' '                                         ADABAS
SQC101=' '                                         ADABAS
SQC701=0                                           ADABAS
SQDR01=188                                         ADABAS
SQCD01=SQDF01                                      ADABAS
SQCC01='S1'                                        ADABAS
CALL ADABAS (                                       ADABAS
1  SQCB01,SQFB01,SQRB01,SQSB01,SQVB01,           ADABAS
1  SQDS01                                         )   ADABAS
  SQLRSP=SQCR01                                     ADABAS
  SQLQTY=SQCQ01                                     ADABAS
  SQLISN=SQCS01                                     ADABAS
  IF (SQCR01 .NE. 0                                ADABAS
1  ) THEN                                          ADABAS
    CALL RESPF (                                    ADABAS
1  SQCB01,SQDD00,SQDA00,SQFB01,SQRB01,SQSB01,   ADABAS
1  SQVB01,SQDB00,SQDC00,SQDE00,SQDG00)         ADABAS
    END IF                                          ADABAS
  SQDJ01=SQCQ01                                     ADABAS
  IF (SQDJ01 .GT. 0) THEN                          ADABAS
    SQEF01=.FALSE.                                 ADABAS
  ELSE                                             ADABAS
    SQEF01=.TRUE.                                  ADABAS
  END IF                                          ADABAS
  IF (SQDJ01 .LT. SQDH01) SQDH01=SQDJ01          ADABAS
  SQDI01=0                                          ADABAS

```

```

C          00000410
*          00000420
*          EXEC ADABAS          00000430
*          FETCH EMPL          00000440
*          END-EXEC            00000450
*          00000460
SQDA00='00000460'          ADABAS
SQDB00(01)='          EXEC ADABAS          '          ADABAS
SQDC00(01)='          '          ADABAS
SQDB00(02)='          FETCH EMPL          '          ADABAS
SQDC00(02)='          '          ADABAS
SQDB00(03)='          END-EXEC          '          ADABAS
SQDC00(03)='          '          ADABAS
SQDG00=03          ADABAS
IF (SQDI01 .EQ. SQDJ01) SQEF01=.TRUE.          ADABAS
IF (.NOT. SQEF01) THEN          ADABAS
SQEF01=.FALSE.          ADABAS
SQC201='N'          ADABAS
SQC101=' '          ADABAS
SQDR01=188          ADABAS
SQCD01=SQDF01          ADABAS
SQCC01='L1'          ADABAS
CALL ADABAS (          ADABAS
1  SQCB01,SQFB01,SQRB01,SQSB01,SQVB01,          ADABAS
1  SQDS01          )          ADABAS
SQLRSP=SQCR01          ADABAS
SQLQTY=SQCQ01          ADABAS
SQLISN=SQCS01          ADABAS
IF (SQCR01 .EQ. 3) THEN          ADABAS
SQEF01=.TRUE.          ADABAS
ELSE          ADABAS
1IF (SQCR01 .NE. 0          ADABAS
1 ) THEN          ADABAS
CALL RESPF (          ADABAS
1  SQCB01,SQDD00,SQDA00,SQFB01,SQRB01,SQSB01,          ADABAS
1  SQVB01,SQDB00,SQDC00,SQDE00,SQDG00)          ADABAS
END IF          ADABAS
END IF          ADABAS
IF (SQEF01) THEN          ADABAS
SQLCOD=003          ADABAS
ELSE          ADABAS
SQLCOD=0          ADABAS
END IF          ADABAS
C          00000470
1 IF (SQLCOD .EQ. 3) GOTO 2          00000480
C          00000490
WRITE (6,20) PID,NAME,FNAME,BIRTH,SEX          00000500
C          00000510
*          00000520
*          EXEC ADABAS          00000530
*          FETCH EMPL          00000540
*          END-EXEC            00000550

```

APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

```

*
SQDA00='00000560'
SQDB00(01)='          EXEC ADABAS          '
SQDC00(01)='          '
SQDB00(02)='          FETCH EMPL          '
SQDC00(02)='          '
SQDB00(03)='          END-EXEC          '
SQDC00(03)='          '
SQDG00=03
IF (SQDI01 .EQ. SQDJ01) SQEF01=.TRUE.
IF (.NOT. SQEF01) THEN
SQEF01=.FALSE.
SQC201='N'
SQC101=' '
SQDR01=188
SQCD01=SQDF01
SQCC01='L1'
CALL ADABAS (
1  SQCB01,SQFB01,SQRB01,SQSB01,SQVB01,
1  SQDS01
)
SQLRSP=SQCR01
SQLQTY=SQCQ01
SQLISN=SQCS01
IF (SQCR01 .EQ. 3) THEN
SQEF01=.TRUE.
ELSE
1IF (SQCR01 .NE. 0
1 ) THEN
CALL RESPF (
1  SQCB01,SQDD00,SQDA00,SQFB01,SQRB01,SQSB01,
1  SQVB01,SQDB00,SQDC00,SQDE00,SQDG00)
END IF
END IF
IF (SQEF01) THEN
SQLCOD=003
ELSE
SQLCOD=0
END IF
C
GOTO 1
C
2 CONTINUE
C
*
EXEC ADABAS
*
CLOSE EMPL
*
END-EXEC
*
SQDA00='00000660'
SQDB00(01)='          EXEC ADABAS          '
SQDC00(01)='          '
SQDB00(02)='          CLOSE EMPL          '

```

```

SQDC00(02)=' ' ADABAS
SQDB00(03)=' END-EXEC ' ADABAS
SQDC00(03)=' ' ADABAS
SQDG00=03 ADABAS
SQC101='I' ADABAS
SQC201='S' ADABAS
SQDR01=188 ADABAS
SQCD01=SQDF01 ADABAS
SQCC01='RC' ADABAS
CALL ADABAS ( ADABAS
1 SQCB01,SQFB01,SQRB01,SQSB01,SQVB01, ADABAS
1 SQDS01 ) ADABAS
SQLRSP=SQCR01 ADABAS
SQLQTY=SQCQ01 ADABAS
SQLISN=SQCS01 ADABAS
IF (SQCR01 .NE. 0 ADABAS
1 ) THEN ADABAS
CALL RESPF ( ADABAS
1 SQCB01,SQDD00,SQDA00,SQFB01,SQRB01,SQSB01, ADABAS
1 SQVB01,SQDB00,SQDC00,SQDE00,SQDG00) ADABAS
END IF ADABAS
C 00000670
* 00000680
* EXEC ADABAS 00000690
* DBCLOSE 00000700
* END-EXEC 00000710
* 00000720
SQDA00='00000720' ADABAS
SQDB00(01)=' EXEC ADABAS ' ADABAS
SQDC00(01)=' ' ADABAS
SQDB00(02)=' DBCLOSE ' ADABAS
SQDC00(02)=' ' ADABAS
SQDB00(03)=' END-EXEC ' ADABAS
SQDC00(03)=' ' ADABAS
SQDG00=03 ADABAS
SQC400=0500 ADABAS
SQC200=' ' ADABAS
SQDR00=188 ADABAS
SQCD00=SQDF00 ADABAS
SQCC00='CL' ADABAS
CALL ADABAS ( ADABAS
1 SQCB00,SQFB00,SQRB00,SQSB00,SQVB00, ADABAS
1 SQDS00 ) ADABAS
IF (SQCR00 .NE. 0 ADABAS
1 ) THEN ADABAS
CALL RESPF ( ADABAS
1 SQCB00,SQDD00,SQDA00,SQFB00,SQRB00,SQSB00, ADABAS
1 SQVB00,SQDB00,SQDC00,SQDE00,SQDG00) ADABAS
END IF ADABAS
C 00000730
10 FORMAT ('1PERSONNEL-ID',8X,'NAME',13X,'FIRST-NAME',8X, 00000740
* 'BIRTH',1X,'SEX' / 1X,64('*') / ) 00000750

```

APPENDIX I - EXAMPLE OF FORTRAN CODE GENERATED BY ADABAS NATIVE SQL

20	FORMAT (3X,A8,3X,A20,1X,A20,1X,A6,1X,A1)	00000760
C		00000770
	END	00000780

17 APPENDIX J - PL/I EXAMPLES

▪ Example 1	336
▪ Example 2	338
▪ Example 3	340

This chapter covers the following topics:

Example 1

```

PEX1 : PROC OPTIONS(MAIN);
/*  AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH
   CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS PERSONNEL-ID
   NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,
   PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE
   FOLLOWING CONDITION:
       PERSONNEL-ID BETWEEN 10000001 AND 19999999
       MODEL-YEAR-MAKE >
       CLASS = 'C'
*/
/*
DCL 1  START_STRUC,
      2  START_MODEL          CHAR(20)      INIT('MERCEDES-BENZ'),
      2  START_YEAR_MAKE     PIC '(2)9'    INIT(86);
DCL   START_MODEL_YEAR_MAKE CHAR(22)      BASED(ADDR(START_STRUC));
/*
DCL 1  HEADER,
      2  FILLER1              CHAR(12)     INIT('PERSONNEL-ID'),
      2  FILLER2              CHAR(8)      INIT(' '),
      2  FILLER3              CHAR(4)      INIT('NAME'),
      2  FILLER4              CHAR(13)     INIT(' '),
      2  FILLER5              CHAR(10)     INIT('FIRST-NAME'),
      2  FILLER6              CHAR(8)      INIT(' '),
      2  FILLER7              CHAR(5)      INIT('BIRTH'),
      2  FILLER8              CHAR(1)      INIT(' '),
      2  FILLER9              CHAR(3)      INIT('SEX');
DCL 1  HEADER2                CHAR(64)     INIT((64)'*');
DCL 1  LINE1,
      2  FILLER1              CHAR(2)      INIT(' '),
      2  PERSONNEL_NR         CHAR(8)      INIT(' '),
      2  FILLER2              CHAR(3)      INIT(' '),
      2  LAST_NAME            CHAR(20)     INIT(' '),
      2  FILLER3              CHAR(1)      INIT(' '),
      2  F_NAME               CHAR(20)     INIT(' '),
      2  FILLER4              CHAR(1)      INIT(' '),
      2  BIRTHDAY             CHAR(6)      INIT(' '),
      2  FILLER5              CHAR(1)      INIT(' '),
      2  KIND                 CHAR(1)      INIT(' ');
/*
          EXEC ADABAS
      BEGIN DECLARE SECTION
      END-EXEC

```



```

/*                                                                 */
      EXEC ADABAS
      DECLARE EMPL CURSOR FOR
      SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX
      FROM EMPLOYEES, VEHICLES
      WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID
            AND PERSONNEL-ID BETWEEN '10000001' AND '19999999'
            AND VEHICLES.MODEL-YEAR-MAKE > :START_MODEL_YEAR_MAKE
            AND VEHICLES.CLASS = 'C'
      END-EXEC
/*                                                                 */

      PUT SKIP EDIT (HEADER) (A);
      PUT SKIP EDIT (HEADER2) (A);
      PUT SKIP;
/*                                                                 */

      EXEC ADABAS
      OPEN EMPL
      END-EXEC
/*                                                                 */

      EXEC ADABAS
      FETCH EMPL
      END-EXEC
/*                                                                 */

      DO WHILE (ADACODE * = 3);
          PERSONNEL_NR = PERSONNEL_ID;
          LAST_NAME = NAME;
          F_NAME = FIRST_NAME;
          BIRTHDAY = BIRTH;
          KIND = SEX;
          PUT SKIP EDIT (LINE1) (A);
              EXEC ADABAS
              FETCH EMPL
              END-EXEC
      END;
/*                                                                 */

      EXEC ADABAS
      CLOSE EMPL
      END-EXEC
/*                                                                 */

      EXEC ADABAS
      DBCLOSE
      END-EXEC
/*                                                                 */
END PEX1;

```

Example 2

```

PEX2 : PROC OPTIONS(MAIN);
/*      DELETE AN EMPLOYEE RECORD AND RELEASE ALL CARS WHICH ARE
      ASSIGNED TO THIS EMPLOYEE. A PRIVATE CARS WILL BE DELETED
      AND A COMPANY CAR WILL BE MADE A POOL-CAR WHICH IS IDENTIFIED
      BY ITS PERSONNEL-ID CONTAINING ONLY THE COUNTRY CODE.
*/
/*
DCL  PERSONNEL_NUMBER      CHAR(8)      INIT ('20007100');
DCL  EMPLOYEE_ISN         FIXED BIN(31) INIT(0);
DCL 1 COUNTRY_NUM,
      2 COUNTRY_NO         CHAR(1)      INIT (' ') ,
      2 FILLER             CHAR(14)     INIT (' ');
DCL  COUNTRY_NUMBER       CHAR(15) BASED(ADDR(COUNTR_NUM));
/*
      EXEC ADABAS
      BEGIN DECLARE SECTION
      END-EXEC
/*
      EXEC ADABAS
      READ LOGICAL
      DECLARE VEH1 CURSOR FOR
      SELECT REG-NUM, PERSONNEL-ID, CLASS
      FROM VEHICLES
      WHERE PERSONNEL-ID GE :PERSONNEL-NUMBER
      OPTIONS HOLD
      ORDER BY PERSONNEL-ID
      END-EXEC
/*
*** FIND EMPLOYEE
      EXEC ADABAS
      FIND
      SELECT
      FROM EMPLOYEES EMPLOYEES_1
      WHERE PERSONNEL-ID = :PERSONNEL_NUMBER
      OPTIONS HOLD
      END-EXEC
/*
***      IF THE PERSONNEL-ID EXISTS DELETE THE EMPLOYEE AND READ THE
***      VEHICLES FILE
      IF EMPLOYEES_1.QUANTITY = 1 THEN
      DO;
          EMPLOYEE_ISN = EMPLOYEES_1.ISN;
          CALL DELETE_EMPLOYEE;
          CALL READ_VEHICLES_FILE;
      */

```

```

        END;
        ELSE
            PUT SKIP EDIT
            ('NO EMPLOYEE FOUND WITH PERSONNEL-ID ',PERSONNEL_NUMBER)(A);
/*
            EXEC ADABAS
            DBCLOSE
            END-EXEC
/*****

```

```

DELETE_EMPLOYEE : PROC;
/*
            EXEC ADABAS
            DELETE
            FROM EMPLOYEES
            WHERE ISN = :EMPLOYEE_ISN
            END-EXEC
/*
            PUT SKIP EDIT
            ('EMPLOYEE ',PERSONNEL_NUMBER,' HAS BEEN DELETED')(A);
/*
END DELETE_EMPLOYEE;
/*****
READ_VEHICLES_FILE : PROC;
/*
            EXEC ADABAS
            OPEN VEH1
            END-EXEC
/*
            EXEC ADABAS
            FETCH VEH1
            END-EXEC
/*
            DO WHILE (ADACODE *= 3 &
                    VEHICLES.PERSONNEL_ID = PERSONNEL_NUMBER);
                IF CLASS = 'P' THEN
                    DO;
                        EXEC ADABAS
                        DELETE
                        FROM VEHICLES
                        WHERE CURRENT OF VEH1
                        END-EXEC
                        PUT SKIP EDIT
                        ('PRIVATE CAR ',REG_NUM,' HAS BEEN DELETED')(A);
                    END;
                ELSE
                    DO;
                        COUNTRY_NUMBER = VEHICLES.PERSONNEL_ID;
                        VEHICLES.PERSONNEL_ID = COUNTRY_NO;
                        EXEC ADABAS
                        UPDATE VEHICLES

```

```

                WHERE CURRENT OF VEH1
                END-EXEC
                PUT SKIP EDIT
                ('COMPANY CAR ',REG_NUM,' HAS BEEN UPDATED')(A);
        END;
/*                                                    */
                EXEC ADABAS
                FETCH VEH1
                END-EXEC
/*                                                    */
        END;
/*                                                    */
                EXEC ADABAS
                CLOSE VEH1
                END-EXEC
/*                                                    */
                EXEC ADABAS
                COMMIT WORK
                END-EXEC
/*                                                    */
END READ_VEHICLES_FILE;
/*                                                    */
END PEX2;

```

Example 3

```

PEX3 : PROC OPTIONS(MAIN);
/*      SALARY INCREASE.
        THIS PROGRAM INCREASES THE SALARY OF EVERY EMPLOYEE BY
        4 PERCENT.
        THE DEPARTMENT, THE OVERALL AMOUNT OF PAY RISE FOR THE
        DEPARTMENT AND THE PAY RISE FOR ALL DEPARTMENTS WILL BE PRINTED
        OUT.
        THE PROGRAM IS RESTARTABLE. AFTER AN ABNORMAL TERMINATION THE
        PROGRAM EXECUTION WOULD RESTART WITH THE LAST DEPARTMENT
        WHOSE SALARY UPDATE HAD BEEN COMPLETED BEFORE THE ABEND
        OCCURED.
*/
/*                                                    */
DCL 1  COMM_DATA,
      2  COMMIT_DEPARTMENT    CHAR(6)          INIT (' '),
      2  COMMIT_SUM          FIXED DEC(10)    INIT (0);
DCL   COMMIT_DATA           CHAR(12) BASED(ADDR(COMM_DATA));
DCL   START_DEPT           CHAR(6)          INIT (' ');
DCL   IND                  FIXED BIN(15)    INIT (0);
DCL   I                    FIXED BIN(15)    INIT (0);
DCL   J                    FIXED BIN(15)    INIT (0);
DCL   NEW_SALARY          FIXED DEC(9)     INIT (0);

```

```

DCL INCREASE          FIXED DEC(9)  INIT (0);
DCL SUM_DEPARTMENT    FIXED DEC(10) INIT (0);
DCL SUM_TOTAL         FIXED DEC(11) INIT (0);
/*                                                             */
DCL 1 HEADER,
      2 FILLER1          CHAR(10)    INIT ('DEPARTMENT'),
      2 FILLER2          CHAR(15)    INIT (' '),
      2 FILLER3          CHAR(15)    INIT ('SALARY INCREASE');
DCL 1 LINE1,
      2 FILLER1          CHAR(3)     INIT (' '),
      2 DEPARTMENT      CHAR(6)     INIT (' '),
      2 FILLER2          CHAR(16)    INIT (' '),
      2 SUM_DEPT        PIC 'Z,ZZZ,ZZZ,ZZ9';
DCL 1 FOOT_LINE,
      2 FILLER1          CHAR(21)    INIT ('TOTAL SALARY INCREASE'),
      2 FILLER           CHAR(3)     INIT (' : '),
      2 TOTAL_SUM_DEPT  PIC 'ZZ,ZZZ,ZZZ,ZZZ';
/*                                                             */
          EXEC ADABAS
          BEGIN DECLARE SECTION
          END-EXEC

```

```

/*                                                             */
          EXEC ADABAS
          HISTOGRAM
          DECLARE EMP1 CURSOR FOR
          SELECT DEPT
          FROM EMPLOYEES EMPLOYEES_1
          WHERE DEPT GE :COMMIT_DEPARTMENT
          GROUP BY DEPT
          END-EXEC
/*                                                             */
          EXEC ADABAS
          READ LOGICAL
          DECLARE EMP2 CURSOR FOR
          SELECT PERSONNEL-ID, DEPT, SALARY, INCOME(COUNT)
          FROM EMPLOYEES
          WHERE DEPT GE :START_DEPT
          OPTIONS HOLD
          ORDER BY DEPT
          END-EXEC
/*                                                             */
          EXEC ADABAS
          CONNECT 'INCREASE'
          UPD=EMPLOYEES
          AND USERDATA INTO :COMMIT_DATA
          END-EXEC
/*
          A HISTOGRAM STATEMENT IS USED TO ASCERTAIN THE NUMBER OF
          EMPLOYEES PER DEPARTMENT
          */

```

```

        EXEC ADABAS
        OPEN EMP1
        END-EXEC
/*                                          */
        EXEC ADABAS
        FETCH EMP1
        END-EXEC
/*                                          */
        IF COMMIT_DATA *=' ' THEN CALL RESTART;
/*                                          */
        START_DEPT = EMPLOYEES_1.DEPT;
/*                                          */
        EXEC ADABAS
        OPEN EMP2
        END-EXEC
/*                                          */
        PUT SKIP EDIT (HEADER) (A);
        PUT SKIP LIST ((40)'*');
        PUT SKIP;
/*                                          */
        DO WHILE (ADACODE *=' 3');
            CALL HIST_EMPL;
        END;

```

```

/*                                          */
        EXEC ADABAS
        CLOSE EMP1
        END-EXEC
/*                                          */
        EXEC ADABAS
        CLOSE EMP2
        END-EXEC
/*                                          */
        PUT SKIP;
        PUT SKIP LIST ((50)'-');
        PUT SKIP;
        TOTAL_SUM_DEPT = SUM_TOTAL;
        PUT SKIP EDIT (FOOT_LINE) (A);
        COMMIT_DATA = '  ';
/*                                          */
        EXEC ADABAS
        DBCLOSE
        USERDATA = :COMMIT_DATA
        END-EXEC
/*****
RESTART : PROC;
        PUT SKIP LIST ('LAST PROGRAM RUN TERMINATED ABNORMALLY');
        PUT SKIP EDIT ('LAST DEPARTMENT WAS: ',COMMIT_DEPARTMENT)(A);
/*                                          */
        EXEC ADABAS
        FETCH EMP1

```

```

                                END-EXEC
END RESTART;
/*****/

```

```

HIST_EMPL : PROC;
/*
    THE EMPLOYEES FILE WILL BE READ UNTIL ALL RECORDS FOR THE
    DEPARTMENT HAVE BEEN PROCESSED AND THE SALARY HAS BEEN
    UPDATED
                                                                */
    DO J=1 BY 1 TO EMPLOYEES_1.QUANTITY;
        EXEC ADABAS
        FETCH EMP2
        END-EXEC
/*
    THE SALARY INCREASE CAN BE EXECUTED WHEN THE COUNT OF THE
    PERIODIC GROUP IS LESS THAN 40.
                                                                */
        IF C_INCOME <= 40 THEN
            CALL SALARY_INCREASE;
        ELSE
            PUT SKIP EDIT
            ('UPDATE PERSON ',PERSONNEL_ID,' NOT POSSIBLE')(A);
    END;
/*
                                                                */
    DEPARTMENT = EMPLOYEES.DEPT;
    SUM_DEPT = SUM_DEPARTMENT;
    SUM_DEPARTMENT = 0;
    PUT SKIP EDIT (LINE1) (A);
/*
                                                                */
    COMMIT_DEPARTMENT = EMPLOYEES.DEPT;
    COMMIT_SUM = SUM_TOTAL;
        EXEC ADABAS
    COMMIT WORK
    USERDATA = :COMMIT_DATA
    END-EXEC
/*
                                                                */
        EXEC ADABAS
    FETCH EMP1
    END-EXEC
/*
                                                                */
END HIST_EMPL;
/*****/
SALARY_INCREASE : PROC;
    INCREASE = SALARY(1) * 0.04;
    NEW_SALARY = SALARY(1) + INCREASE;
    IND = C_INCOME + 1;
/*
                                                                */
    DO I=C_INCOME BY -1 TO 0;
        SALARY(IND) = SALARY(I);
        IND = IND - 1;
/*
                                                                */
    END;
/*
                                                                */

```

```
        SALARY(1) = NEW_SALARY;
/*                                          */
        EXEC ADABAS
        UPDATE EMPLOYEES
        WHERE CURRENT OF EMP2
        END-EXEC
/*                                          */
        SUM_DEPARTMENT = SUM_DEPARTMENT + INCREASE;
        SUM_TOTAL = SUM_TOTAL + INCREASE;
END SALARY_INCREASE;
/*                                          */
END PEX3;
```


18 APPENDIX - EXAMPLE OF PL/I CODE GENERATED BY ADABAS NATIVE SQL

```
PEX1 : PROC OPTIONS(MAIN);                                00000010
/* AN EXAMPLE OF SOFT COUPLING WITH A SEARCH CRITERION WHICH 00000020
CONTAINS FIELDS TAKEN FROM TWO FILES. THE FIELDS PERSONNEL-ID 00000030
NAME, FIRST-NAME, BIRTH AND SEX (FROM THE MAIN FILE,          00000040
PERSONNEL-ID) ARE PRINTED FOR RECORDS THAT SATISFY THE        00000050
FOLLOWING CONDITION:                                          00000060
    PERSONNEL-ID BETWEEN 10000001 AND 19999999                00000070
    MODEL-YEAR-MAKE >                                         00000080
    CLASS = 'C'                                               */00000090
/*                                                            */00000100
DCL 1 START_STRUC,                                           00000110
    2 START_MODEL      CHAR(20)    INIT('MERCEDES-BENZ'),      00000120
    2 START_YEAR_MAKE  PIC '(2)9'  INIT(86);                   00000130
DCL  START_MODEL_YEAR_MAKE CHAR(22)  BASED(ADDR(START_STRUC));00000140
/*                                                            */00000150
DCL 1 HEADER,                                               00000160
    2 FILLER1          CHAR(12)    INIT('PERSONNEL-ID'),      00000170
    2 FILLER2          CHAR(8)     INIT(' '),                  00000180
    2 FILLER3          CHAR(4)     INIT('NAME'),                00000190
    2 FILLER4          CHAR(13)    INIT(' '),                  00000200
    2 FILLER5          CHAR(10)    INIT('FIRST-NAME'),         00000210
    2 FILLER6          CHAR(8)     INIT(' '),                  00000220
    2 FILLER7          CHAR(5)     INIT('BIRTH'),               00000230
    2 FILLER8          CHAR(1)     INIT(' '),                  00000240
    2 FILLER9          CHAR(3)     INIT('SEX');                 00000250
DCL 1 HEADER2          CHAR(64)    INIT((64)'*');              00000260
DCL 1 LINE1,                                                 00000270
    2 FILLER1          CHAR(2)     INIT(' '),                  00000280
    2 PERSONNEL_NR     CHAR(8)     INIT(' '),                  00000290
    2 FILLER2          CHAR(3)     INIT(' '),                  00000300
    2 LAST_NAME        CHAR(20)    INIT(' '),                  00000310
    2 FILLER3          CHAR(1)     INIT(' '),                  00000320
```

APPENDIX - EXAMPLE OF PL/I CODE GENERATED BY ADABAS NATIVE SQL

```

2 F_NAME          CHAR(20)    INIT(' '),          00000330
2 FILLER4         CHAR(1)     INIT(' '),          00000340
2 BIRTHDAY       CHAR(6)     INIT(' '),          00000350
2 FILLER5        CHAR(1)     INIT(' '),          00000360
2 KIND           CHAR(1)     INIT(' ');          00000370
/*                                                       */00000380
-/*                                                       ** 00000390
          EXEC ADABAS                                00000400
          BEGIN DECLARE SECTION                       00000410
          END-EXEC                                    00000420
**                                                       */ 00000430
DCL ADACODE FIXED BIN(15) INIT (0);                  ADABAS
DCL ADABAS   ENTRY OPTIONS(ASM,INTER);              ADABAS
DCL RESPINT  ENTRY OPTIONS(ASM,INTER);              ADABAS
DCL 1 CONTROL_BLOCKOPN UNAL,                        ADABAS
   3 FILLER10PN CHAR(2)      INIT ('AS') ,          ADABAS
   3 COMMAND_CODEOPN CHAR(2) ,                      ADABAS
   3 COMMAND_IDOPN CHAR(4)   INIT ('OPEN') ,        ADABAS
   3 FILE_NUMBEROPN FIXED BIN(15) INIT ( 0) ,      ADABAS
   3 RESPONSE_CODEOPN FIXED BIN(15) INIT (0) ,     ADABAS
   3 ISNOPN FIXED BIN(31) INIT (0) ,               ADABAS
   3 ISN_LOWER_LIMITOPN FIXED BIN(31) INIT (0) ,  ADABAS
   3 ISN_QUANTITYOPN FIXED BIN(31) ,              ADABAS
   3 FORMAT_BUFFER_LENGTHOPN FIXED BIN(15) INIT ( 0) , ADABAS
   3 RECORD_BUFFER_LENGTHOPN FIXED BIN(15) INIT ( 0) , ADABAS
   3 SEARCH_BUFFER_LENGTHOPN FIXED BIN(15) INIT ( 0) , ADABAS
   3 VALUE_BUFFER_LENGTHOPN FIXED BIN(15) INIT ( 0) , ADABAS
   3 ISN_BUFFER_LENGTHOPN FIXED BIN(15) INIT ( 4) , ADABAS
   3 COMMAND_OPTION_10PN CHAR(1) INIT (' ') ,     ADABAS
   3 COMMAND_OPTION_20PN CHAR(1) INIT (' ') ,     ADABAS
   3 ADDITIONS_10PN CHAR(8) INIT (' ') ,          ADABAS
   3 ADDITIONS_20PN CHAR(4) INIT (' ') ,          ADABAS
   3 ADDITIONS_30PN CHAR(8) INIT (' ') ,          ADABAS
   3 ADDITIONS_40PN CHAR(8) INIT (' ') ,          ADABAS
   3 ADDITIONS_50PN ,                               ADABAS
   4 ADDITIONS_5_BNOPN FIXED BIN(31) INIT (0) ,   ADABAS
   4 ADDITIONS_5_580PN CHAR(4) ,                  ADABAS
   3 COMMAND_TIMEOPN FIXED BIN(31) ,              ADABAS
   3 USER_AREAOPN CHAR(4) INIT ('AS') ;          ADABAS
DCL CONTROL_BLOCK_10PN CHAR(80)                    ADABAS
   BASED(ADDR(CONTROL_BLOCKOPN ));                ADABAS
DCL ADDITIONS_1_120PN CHAR(2) DEF ADDITIONS_10PN ; ADABAS
DCL ADDITIONS_1_BNOPN FIXED BIN(15) UNAL          ADABAS
   BASED (ADDR(ADDITIONS_10PN ));                 ADABAS
DCL ADDITIONS_1_580PN CHAR(4) DEF ADDITIONS_10PN POS(5); ADABAS
DCL 1 ADDITIONS_5_DEFOPN BASED (ADDR(ADDITIONS_50PN)), ADABAS
   2 ADDITIONS_5_10PN CHAR(1),                    ADABAS
   2 ADDITIONS_5_280PN CHAR(7);                   ADABAS
DCL 1 ADDITIONS_4_DEFOPN BASED (ADDR(ADDITIONS_40PN)), ADABAS
   2 ADDITIONS_4_120PN FIXED BIN(15),             ADABAS
   2 ADDITIONS_4_340PN FIXED BIN(15),             ADABAS
   2 ADDITIONS_4_560PN FIXED BIN(15),             ADABAS

```

```

2 ADDITIONS_4_780PN FIXED BIN(15); ADABAS
DCL DBIDOPN CHAR(1) BASED (ADDR(FILE_NUMBEROPN )); ADABAS
DCL FORMAT_BUFOPN CHAR (0001) ; ADABAS
DCL SEARCH_BUFOPN CHAR (0001) ; ADABAS
DCL VALUE_BUFOPN CHAR (0001) ; ADABAS
DCL ISN_BUFOPN CHAR (0001) ; ADABAS
DCL RECORD_BUFOPN CHAR (1500) ; ADABAS
DCL OPENTYPE CHAR (0010) ; ADABAS
DCL DDFILE PIC'999' INIT ( 30) ; ADABAS
DCL CSEQ CHAR(8) ; ADABAS
DCL CLN1(20) CHAR(40) ; ADABAS
DCL CLN2(20) CHAR(40) ; ADABAS
DCL TRCE CHAR(7) ; ADABAS
DCL CLNNUM FIXED BIN(15) ; ADABAS
DCL SQLRSP FIXED BIN(15) ; ADABAS
DCL SQLQTY FIXED BIN(31) ; ADABAS
DCL SQLISN FIXED BIN(31) ; ADABAS
DCL SAVE_DBID_10PN FIXED BIN(15) ; ADABAS
DCL 1 SAVE_DBID_DEFOPN BASED(ADDR(SAVE_DBID_10PN )), ADABAS
2 FILLEROPN CHAR(1), ADABAS
2 SAVE_DBIDOPN CHAR(1); ADABAS
DCL 1 FORMAT_BUFEMPL_1 , ADABAS
2 FILLE001 CHAR(34) INIT('AA,8,A,AE,20,A,AC,20,A,AH,6,U,AG,1'), ADABAS
2 FILLE002 CHAR(03) INIT(',A. '), ADABAS
FORMAT_BUFEMPL CHAR(00037) ADABAS
BASED (ADDR( FORMAT_BUFEMPL_1 )); ADABAS
DCL 1 SEARCH_BUFEMPL_1 , ADABAS
2 FILLE001 CHAR(34) INIT('(22,AA,24,AC)/22/AA,8,A,S,AA,8,A,D'), ADABAS
2 FILLE002 CHAR(25) INIT(',/24/A0,22,A,GT,D,AH,1,A. '), ADABAS
SEARCH_BUFEMPL CHAR(00059) ADABAS
BASED (ADDR( SEARCH_BUFEMPL_1 )); ADABAS
DCL 1 EMPLOYEES UNAL, ADADATA
2 RECORD_BUFEMPL_1 , ADADATA
3 PERSONNEL_ID CHAR (0008) , ADADATA
3 NAME CHAR (0020) , ADADATA
3 FIRST_NAME CHAR (0020) , ADADATA
3 BIRTH PIC '(0005)99' , ADADATA
3 SEX CHAR (0001) , ADADATA
2 ISN FIXED BIN(31), ADADATA
2 QUANTITY FIXED BIN(31), ADADATA
2 RESPONSE_CODE FIXED BIN(15), ADADATA
RECORD_BUFEMPL CHAR(00055) ADADATA
BASED (ADDR( RECORD_BUFEMPL_1 )); ADADATA
DCL 1 VALUE_BUFEMPL_1 UNAL, ADABAS
2 V_PERSONNEL_ID_F CHAR (0008) ADABAS
INIT(' '), ADABAS
2 V_PERSONNEL_ID_T CHAR (0008) ADABAS
INIT(' '), ADABAS
2 V_MODEL_YEAR_MAKE, ADABAS
3 S_YEAR PIC '(0001)99' ADABAS
INIT(0), ADABAS
3 S_MAKE CHAR (0020) ADABAS

```

```

                INIT(' '), ADABAS
2 V_CLASS CHAR (0001) ADABAS
                INIT(' '), ADABAS
                VALUE_BUFEMPL CHAR(00039) ADABAS
                BASED (ADDR(VALUE_BUFEMPL_1 )); ADABAS
DCL V_MODEL_YEAR_MAKE_EMPL CHAR (0022) ADABAS
    BASED (ADDR( ADABAS
                VALUE_BUFEMPL_1.V_MODEL_YEAR_MAKE )); ADABAS
DCL ISN_BUFEMPL ( 1) FIXED BIN(31); ADABAS
DCL 1 CONTROL_BLOCKEMPL UNAL, ADABAS
3 FILLER1EMPL CHAR(2) INIT ('AS') , ADABAS
3 COMMAND_CODEEMPL CHAR(2) , ADABAS
3 COMMAND_IDEMPL CHAR(4) INIT ('EMPL') , ADABAS
3 FILE_NUMBEREMPL FIXED BIN(15) INIT ( 22) , ADABAS
3 RESPONSE_CODEEMPL FIXED BIN(15) INIT (0) , ADABAS
3 ISNEMPL FIXED BIN(31) INIT (0) , ADABAS
3 ISN_LOWER_LIMITEMPL FIXED BIN(31) INIT (0) , ADABAS
3 ISN_QUANTITYEMPL FIXED BIN(31) , ADABAS
3 FORMAT_BUFFER_LENGTHEMPL FIXED BIN(15) INIT ( 37) , ADABAS
3 RECORD_BUFFER_LENGTHEMPL FIXED BIN(15) INIT ( 55) , ADABAS
3 SEARCH_BUFFER_LENGTHEMPL FIXED BIN(15) INIT ( 59) , ADABAS
3 VALUE_BUFFER_LENGTHEMPL FIXED BIN(15) INIT ( 39) , ADABAS
3 ISN_BUFFER_LENGTHEMPL FIXED BIN(15) INIT ( 4) , ADABAS
3 COMMAND_OPTION_1EMPL CHAR(1) INIT (' ') , ADABAS
3 COMMAND_OPTION_2EMPL CHAR(1) INIT (' ') , ADABAS
3 ADDITIONS_1EMPL CHAR(8) INIT (' ') , ADABAS
3 ADDITIONS_2EMPL CHAR(4) INIT (' ') , ADABAS
3 ADDITIONS_3EMPL CHAR(8) INIT (' ') , ADABAS
3 ADDITIONS_4EMPL CHAR(8) INIT (' ') , ADABAS
3 ADDITIONS_5EMPL , ADABAS
4 ADDITIONS_5_BNEMPL FIXED BIN(31) INIT (0) , ADABAS
4 ADDITIONS_5_58EMPL CHAR(4) , ADABAS
3 COMMAND_TIMEEMPL FIXED BIN(31) , ADABAS
3 USER_AREEMPL CHAR(4) INIT ('AS') ; ADABAS
DCL CONTROL_BLOCK_1EMPL CHAR(80) ADABAS
    BASED(ADDR(CONTROL_BLOCKEMPL)); ADABAS
DCL ADDITIONS_1_12EMPL CHAR(2) DEF ADDITIONS_1EMPL; ADABAS
DCL ADDITIONS_1_BNEMPL FIXED BIN(15) UNAL ADABAS
    BASED (ADDR(ADDITIONS_1EMPL)); ADABAS
DCL ADDITIONS_1_58EMPL CHAR(4) DEF ADDITIONS_1EMPL POS(5); ADABAS
DCL 1 ADDITIONS_5_DEFEMPL BASED (ADDR(ADDITIONS_5EMPL)), ADABAS
    2 ADDITIONS_5_1EMPL CHAR(1), ADABAS
    2 ADDITIONS_5_28EMPL CHAR(7); ADABAS
DCL DBIDEMPL CHAR(1) BASED (ADDR(FILE_NUMBEREMPL)); ADABAS
DCL ISNSIZEEMPL FIXED BIN(31) ; ADABAS
DCL ISNMOREEMPL FIXED BIN(31) ; ADABAS
DCL ISNINDEEMPL FIXED BIN(15) ; ADABAS
DCL SAVE_DBID_1EMPL FIXED BIN(15) ; ADABAS
DCL 1 SAVE_DBID_DEFEMPL BASED(ADDR(SAVE_DBID_1EMPL)), ADABAS
    2 FILLEREMPL CHAR(1), ADABAS
    2 SAVE_DBIDEMPL CHAR(1); ADABAS
DCL EOFEMPL BIT(1) INIT ('O'B); ADABAS

```

```

/*                                                    */00000440
-/*                                                    ** 00000450
      EXEC ADABAS                                     00000460
      DECLARE EMPL CURSOR FOR                         00000470
      SELECT PERSONNEL-ID, NAME, FIRST-NAME, BIRTH, SEX 00000480
      FROM EMPLOYEES, VEHICLES                       00000490
      WHERE EMPLOYEES.PERSONNEL-ID = VEHICLES.PERSONNEL-ID 00000500
            AND PERSONNEL-ID BETWEEN '10000001' AND '19999999' 00000510
            AND VEHICLES.MODEL-YEAR-MAKE > :START_MODEL_YEAR_MAKE 00000520
            AND VEHICLES.CLASS = 'C'                 00000530
      END-EXEC                                       00000540
**                                                    */ 00000550
/*                                                    */00000560
      PUT SKIP EDIT (HEADER) (A);                   00000570
      PUT SKIP EDIT (HEADER2) (A);                  00000580
      PUT SKIP;                                     00000590
/*                                                    */00000600
-/*                                                    ** 00000610
      EXEC ADABAS                                     00000620
      OPEN EMPL                                       00000630
      END-EXEC                                       00000640
**                                                    */ 00000650
      VALUE_BUFEMPL_1.V_PERSONNEL_ID_F = '10000001'; ADABAS
      VALUE_BUFEMPL_1.V_PERSONNEL_ID_T = '19999999'; ADABAS
      V_MODEL_YEAR_MAKE_EMPL = START_MODEL_YEAR_MAKE; ADABAS
      VALUE_BUFEMPL_1.V_CLASS = 'C';                ADABAS
DO;                                                    ADABAS
ISNSIZEEMPL=ISN_BUFFER_LENGTHEMPL/4;                ADABAS
ISNINDEEMPL=1;                                       ADABAS
ISN_LOWER_LIMITEMPL=0;                               ADABAS
COMMAND_OPTION_1EMPL=' ';                            ADABAS
COMMAND_OPTION_2EMPL=' ';                            ADABAS
ISN_BUFFER_LENGTHEMPL=0;                             ADABAS
ISN_QUANTITYEMPL=0                                  ; ADABAS
COMMAND_CODEEMPL='S1';                               ADABAS
CALL ADABAS (                                        ADABAS
      CONTROL_BLOCKEMPL,FORMAT_BUFEMPL,RECORD_BUFEMPL, ADABAS
      SEARCH_BUFEMPL,VALUE_BUFEMPL,                 ADABAS
      ISN_BUFEMPL                                   ); ADABAS
EMPLOYEES.RESPONSE_CODE                             =RESPONSE_CODEEMPL; ADABAS
EMPLOYEES.QUANTITY                                 =ISN_QUANTITYEMPL; ADABAS
EMPLOYEES.ISN                                      =ISNEMPL;      ADABAS
IF RESPONSE_CODEEMPL*≠0                             ADABAS
  THEN DO;                                          ADABAS
CSEQ='00000650'; ADABAS
CLN1(01)='          EXEC ADABAS                    '; ADABAS
CLN2(01)='          ';                               ADABAS
CLN1(02)='          OPEN EMPL                       '; ADABAS
CLN2(02)='          ';                               ADABAS
CLN1(03)='          END-EXEC                         '; ADABAS
CLN2(03)='          ';                               ADABAS
CLNNUM=03;                                          ADABAS

```

```

CALL RESPINT                                ADABAS
  (CONTROL_BLOCKEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
  RECORD_BUFEMPL,SEARCH_BUFEMPL,VALUE_BUFEMPL,
  CLN1,CLN2,TRCE,CLNNUM);                    ADABAS
END;                                          ADABAS
ISNMOREEMPL=ISN_QUANTITYEMPL;                ADABAS
IF ISNMOREEMPL > 0 THEN EOFEMPL= '0'B;        ADABAS
      ELSE EOFEMPL= '1'B;                    ADABAS
IF ISNMOREEMPL<ISNSIZEEMPL THEN ISNSIZEEMPL=ISNMOREEMPL; ADABAS
ISNINDEEMPL=0;                               ADABAS
END;                                          ADABAS
/*                                           */00000660
-/*                                         ** 00000670
      EXEC ADABAS                            00000680
      FETCH EMPL                             00000690
      END-EXEC                               00000700
**                                           */ 00000710
DO;                                          ADABAS
IF ISNINDEEMPL=ISNMOREEMPL THEN EOFEMPL='1'B; ADABAS
IF *EOFEMPL THEN DO;                       ADABAS
EOFEMPL='0'B;                               ADABAS
COMMAND_OPTION_2EMPL='N';                   ADABAS
COMMAND_OPTION_1EMPL=' ';                   ADABAS
COMMAND_CODEEMPL='L1';                       ADABAS
CALL ADABAS (                               ADABAS
  CONTROL_BLOCKEMPL,FORMAT_BUFEMPL,RECORD_BUFEMPL,
  SEARCH_BUFEMPL,VALUE_BUFEMPL,
  ISN_BUFEMPL                               ); ADABAS
EMPLOYEES.RESPONSE_CODE                     =RESPONSE_CODEEMPL; ADABAS
EMPLOYEES.QUANTITY                          =ISN_QUANTITYEMPL; ADABAS
EMPLOYEES.ISN                               =ISNEMPL;          ADABAS
IF RESPONSE_CODEEMPL=3                      ADABAS
THEN EOFEMPL='1'B;                          ADABAS
ELSE                                         ADABAS
IF RESPONSE_CODEEMPL*=0                    ADABAS
  THEN DO;                                  ADABAS
CSEQ='00000710';                            ADABAS
CLN1(01)='          EXEC ADABAS            '; ADABAS
CLN2(01)='          ';                      ADABAS
CLN1(02)='          FETCH EMPL             '; ADABAS
CLN2(02)='          ';                      ADABAS
CLN1(03)='          END-EXEC               '; ADABAS
CLN2(03)='          ';                      ADABAS
CLNNUM=03;                                  ADABAS
  CALL RESPINT                                ADABAS
    (CONTROL_BLOCKEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
    RECORD_BUFEMPL,SEARCH_BUFEMPL,VALUE_BUFEMPL,
    CLN1,CLN2,TRCE,CLNNUM);                  ADABAS
  END;                                        ADABAS
END;                                        ADABAS
END;                                        ADABAS
IF EOFEMPL THEN ADACODE = 003;              ADABAS

```

```

ELSE ADACODE = 0;                                ADABAS
/*                                                */00000720
DO WHILE (ADACODE * = 3);                          00000730
    PERSONNEL_NR = PERSONNEL_ID;                    00000740
    LAST_NAME = NAME;                               00000750
    F_NAME = FIRST_NAME;                            00000760
    BIRTHDAY = BIRTH;                               00000770
    KIND = SEX;                                     00000780
    PUT SKIP EDIT (LINE1) (A);                      00000790
-/*                                                ** 00000800
    EXEC ADABAS                                     00000810
    FETCH EMPL                                       00000820
    END-EXEC                                         00000830
**                                                */ 00000840
DO;                                                  ADABAS
IF ISNINDEEMPL=ISNMOREEMPL THEN EOFEMPL='1'B;     ADABAS
IF *EOFEMPL THEN DO;                               ADABAS
EOFEMPL='0'B;                                       ADABAS
COMMAND_OPTION_2EMPL='N';                           ADABAS
COMMAND_OPTION_1EMPL=' ';                           ADABAS
COMMAND_CODEEMPL='L1';                              ADABAS
CALL ADABAS (                                        ADABAS
    CONTROL_BLOCKEMPL,FORMAT_BUFEMPL,RECORD_BUFEMPL, ADABAS
    SEARCH_BUFEMPL,VALUE_BUFEMPL,                  ADABAS
    ISN_BUFEMPL                                     ); ADABAS
EMPLOYEES.RESPONSE_CODE                             =RESPONSE_CODEEMPL; ADABAS
EMPLOYEES.QUANTITY                                  =ISN_QUANTITYEMPL; ADABAS
EMPLOYEES.ISN                                       =ISNEMPL;          ADABAS
IF RESPONSE_CODEEMPL=3                               ADABAS
THEN EOFEMPL='1'B;                                   ADABAS
ELSE                                                  ADABAS
IF RESPONSE_CODEEMPL* =0                             ADABAS
THEN DO;                                             ADABAS
CSEQ='00000840';                                     ADABAS
CLN1(01)='          EXEC ADABAS          ';        ADABAS
CLN2(01)='          ';                              ADABAS
CLN1(02)='          FETCH EMPL          ';         ADABAS
CLN2(02)='          ';                              ADABAS
CLN1(03)='          END-EXEC          ';           ADABAS
CLN2(03)='          ';                              ADABAS
CLNNUM=03;                                           ADABAS
CALL RESPINT                                         ADABAS
    (CONTROL_BLOCKEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL, ADABAS
    RECORD_BUFEMPL,SEARCH_BUFEMPL,VALUE_BUFEMPL,   ADABAS
    CLN1,CLN2,TRCE,CLNNUM);                          ADABAS
END;                                                 ADABAS
END;                                                 ADABAS
END;                                                 ADABAS
IF EOFEMPL THEN ADACODE = 003;                      ADABAS
ELSE ADACODE = 0;                                    ADABAS
END;                                                 00000850
/*                                                */00000860

```

APPENDIX - EXAMPLE OF PL/I CODE GENERATED BY ADABAS NATIVE SQL

```

- /*
EXEC ADABAS
CLOSE EMPL
END-EXEC
**
**
DO;
COMMAND_OPTION_1EMPL='I';
COMMAND_OPTION_2EMPL='S';
COMMAND_CODEEMPL='RC';
CALL ADABAS (
CONTROL_BLOCKEMPL,FORMAT_BUFEMPL,RECORD_BUFEMPL,
SEARCH_BUFEMPL,VALUE_BUFEMPL,
ISN_BUFEMPL
);
EMPLOYEES.RESPONSE_CODE=RESPONSE_CODEEMPL;
EMPLOYEES.QUANTITY=ISN_QUANTITYEMPL;
EMPLOYEES.ISN=ISNEMPL;
IF RESPONSE_CODEEMPL*≠0
THEN DO;
CSEQ='00000910';
CLN1(01)=' EXEC ADABAS ';
CLN2(01)=' ';
CLN1(02)=' CLOSE EMPL ';
CLN2(02)=' ';
CLN1(03)=' END-EXEC ';
CLN2(03)=' ';
CLNNUM=03;
CALL RESPINT
(CONTROL_BLOCKEMPL,DDFILE,CSEQ,FORMAT_BUFEMPL,
RECORD_BUFEMPL,SEARCH_BUFEMPL,VALUE_BUFEMPL,
CLN1,CLN2,TRCE,CLNNUM);
END;
END;
/*
- /*
EXEC ADABAS
DBCLOSE
END-EXEC
**
**
DO;
RECORD_BUFFER_LENGTHOPN=1500;
COMMAND_OPTION_2OPN=' ';
COMMAND_CODEOPN='CL';
CALL ADABAS (
CONTROL_BLOCKOPN,FORMAT_BUFOPN,RECORD_BUFOPN,
SEARCH_BUFOPN,VALUE_BUFOPN,
ISN_BUFOPN
);
IF RESPONSE_CODEOPN *≠0
THEN DO;
CSEQ='00000970';
CLN1(01)=' EXEC ADABAS ';
CLN2(01)=' ';
CLN1(02)=' DBCLOSE ';

```



```

CLN2(02)='          ' ; ADABAS
CLN1(03)='          END-EXEC ' ; ADABAS
CLN2(03)='          ' ; ADABAS
CLNNUM=03; ADABAS
  CALL RESPINT ADABAS
    (CONTROL_BLOCKOPN ,DDFILE,CSEQ,FORMAT_BUFOPN , ADABAS
      RECORD_BUFOPN ,SEARCH_BUFOPN ,VALUE_BUFOPN , ADABAS
      CLN1,CLN2,TRCE,CLNNUM); ADABAS
  END; ADABAS
END; ADABAS
/* ADABAS */00000980

```

