

OVERVIEW OF STATEMENTS

This chapter covers the following topics:

- Syntax
 - Overview of Adabas Native SQL Statements
 - Database Query Statements
 - Data Storage READ Statements
 - Associator READ Statement
 - Statements for Processing Multiple Records
 - Database Modification Statements
 - Logical Transaction Processing Statements
 - Checkpointing Statement
 - Other Adabas Native SQL Statements
 - Adabas Native SQL Clauses
-

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.

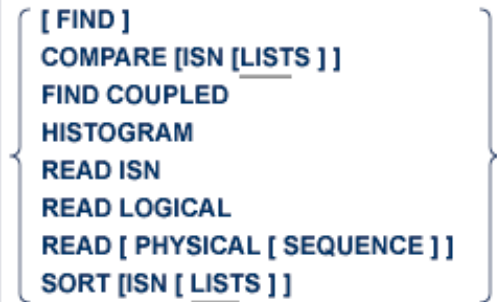
Syntax Diagram for Adabas Native SQL Data Retrieval Statements

```

EXEC ADABAS
  statement-name
  [ DECLARE cursor-name CURSOR [FOR] ]
  [ SELECT { field-name,... | * } ]
  FROM { file [ alias ] },...
  [ WHERE search-criterion ]
  [ OPTIONS
    [ {
      AUTODBD
      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*



```
[ FIND ]  
COMPARE [ISN [LISTS ]]  
FIND COUPLED  
HISTOGRAM  
READ ISN  
READ LOGICAL  
READ [ PHYSICAL [ SEQUENCE ]]  
SORT [ISN [LISTS ]]
```

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>
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.

Statement	Action
CONNECT	<p>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).</p> <p>This statement generates the Adabas command OP (Open User Session).</p>
COPY	<p>Permits a file layout generated by Predict as Ada, COBOL, FORTRAN or PL/I code to be copied into the program.</p>
DBCLOSE	<p>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.</p> <p>This statement generates the Adabas command CL (Close User Session).</p>
GENERATE	<p>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.</p>
HOLD	<p>Places a record in hold status. Other programs cannot interfere with this record so long as it is in hold status.</p> <p>A record must be put in hold status before it can be deleted or updated.</p> <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>

Statement	Action
SAVE	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).
TRACE	A debugging aid used to switch trace printing of all executed Adabas Native SQL statements on and off.
WHENEVER	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.
WRITE TO LOG	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. This statement generates the Adabas command C5 (Write User Data To Protection Log).

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 `SELECT` keyword followed by a curly brace containing the text `field-name , ...`. A small dot is positioned below the comma.

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_BUFPER 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_BUFPER;
```

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 INSPES
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 {
  {
    AUTOBID
    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.

AUTOBID 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 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 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, FORTRAN 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.