

Multi-Fetch Clause

This section covers the multi-fetch record retrieval functionality for Adabas databases. This feature is available both under Windows/UNIX and on mainframes, however, there are differences in the usage of multi-fetch on those systems.

Note:

The multi-fetch functionality is only supported for Adabas.

- Multi-Fetch under Windows and UNIX
- Multi-Fetch on Mainframes

Multi-Fetch under Windows and UNIX

By default, Natural uses single-fetch to retrieve data from Adabas databases. This default can be configured using the profile parameter MFSET.

Values "ON" (multi-fetch) and "OFF" (single-fetch) define the default behavior. If MFSET is set to "NEVER", Natural always uses single-fetch mode and ignores any settings at statement level.

Multi-fetch processing is supported for the statements FIND, READ and HISTOGRAM that do not involve database modification. To minimize the number of required Adabas calls, several results are retrieved in one call.

If nested database loops that refer to the same Adabas file contain UPDATE statements in one of the inner loops, Natural continues processing the outer loops with the updated values. This implies in multi-fetch mode, that an outer logical READ loop has to be repositioned if an inner database loop updates the value of the descriptor that is used for sequence control in the outer loop. If this attempt leads to a conflict for the current descriptor, an error is returned. To avoid this situation, we recommend that you disable multi-fetch in the outer database loops.

In general, multi-fetch mode improves performance when accessing Adabas databases. In some cases, however, it might be advantageous to use single-fetch to enhance performance, especially if database modifications are involved.

The default processing mode can be overridden at statement level. For further information on the syntax, see the READ, FIND or HISTOGRAM statements, subsection MULTI-FETCH Clause.

Multi-Fetch on Mainframes

In standard mode, Natural does not read multiple records with a single database call; it always operates in a one-record-per-fetch mode. This kind of operation is solid and stable, but can take some time if a large number of database records are being processed.

To improve the performance of those programs, you can apply the Multi-Fetch clause, that allows you to define the Multi-Fetch-Factor, a numeric value that specifies the number of records read per database access.

<pre> { FIND READ HISTOGRAM } </pre>	<pre> MULTI-FETCH [OF] <multi-fetch-factor> </pre>
--	--

<multi-fetch-factor>

The <multi-fetch-factor> is either a constant or a variable with a format integer (I4).

At statement execution time, the runtime checks if a <multi-fetch-factor> greater than 1 is supplied for the database statement.

If the <multi-fetch-factor> is less than or equal to 1:

the database call is continued in the usual one-record-per-access mode.

If the <multi-fetch-factor> is greater than 1:

the database call is prepared dynamically to read multiple records (e.g. 10) with a single database access into an auxiliary buffer (multi-fetch buffer). If successful, the first record is transferred into the underlying data view. Upon the execution of the next loop, the data view is filled directly from the multi-fetch buffer, without database access. After all records are fetched from the multi-fetch buffer, the next loop results in the next record set being read from the database. If the database loop is terminated (either by end-of-records, ESCAPE, STOP, etc.) the content of the multi-fetch buffer is released.

Considerations for Multi-Fetch Usage

- A multi-fetch access is only supported for a browse loop; in other words, when the records are read with "no hold".
- The program does not receive "fresh" records from the database for every loop, but operates with images retrieved at the most recent multi-fetch access.
- If a loop repositioning is triggered for a READ / HISTOGRAM statement, the content of the multi-fetch buffer at that point is released.
- If a dynamic direction change (IN DYNAMIC...SEQUENCE) is coded for a READ / HISTOGRAM statement, the multi-fetch feature is not possible and leads to a corresponding syntax error at compilation.
- The first record of a FIND loop is retrieved with the initial S1 command. Since Adabas multi-fetch is just defined for all kinds of Lx commands, it first can be used from the second record.
- The size occupied by a database loop in the multi-fetch buffer is determined according to the rule:

$$\begin{aligned} & ((\text{record-buffer-length} + \text{isn-buffer-entry-length}) * \text{multi-fetch-factor}) + 4 + \text{header-length} \\ & = \\ & ((\text{size-of-view-fields} + 20) * \text{multi-fetch-factor}) + 4 + 128 \end{aligned}$$

In order to keep the required space small, the multi-fetch factor is automatically reduced at runtime, if

- the "loop-limit" (e.g. READ (2) ..) is smaller, but only if no WHERE clause is involved;
- the "ISN quantity" (for FIND statement only) is smaller;
- the resulting size of the Record-Buffer or ISN-Buffer exceeds 32KB.

Moreover, the multi-fetch option is completely ignored at runtime, if

- the multi-fetch factor contains a value less equal 1;
- the multi-fetch buffer is not available or does not have enough free space (for more details, refer to the section below).

Size of the Multi-Fetch Buffer

In order to control the amount of storage available for multi-fetch purposes, you can limit the maximum size of the multi-fetch buffer.

Inside the NATPARAM definition, you can make a static assignment via the profile parameter

NTDS MULFETCH,nn

At session start, you can also use the dynamic profile parameter

DS=(MULFETCH,nn)

where "nn" represents the complete size allowed to be allocated for multi-fetch purposes (in KB). The value may be set in the range (0 - 1024), with a default value of 64. Setting a high value does not necessarily mean having a buffer allocated of that size, since the multi-fetch handler makes dynamic allocations and resizes, depending on what is really needed to execute a multi-fetch database statement. If no multi-fetch database statement is executed in a Natural session, the multi-fetch buffer will never be created, regardless of which value was set.

If value 0 is specified, the multi-fetch processing is completely disabled, no matter if a database statement contains a "MULTI-FETCH OF .." clause or not. This allows to completely switch off all multi-fetch activities when there is not enough storage available in the current environment or for debugging purposes.

Note:

Due to existing Adabas limitations, you may not have a Record-Buffer or ISN-Buffer larger than 32KB.

Therefore you need only a maximum of 64KB space in the multi-fetch buffer for a single

READ/FIND/HISTOGRAM loop. The required value setting for the multi-fetch buffer depends on the number of nested database loops you want to serve with multi-fetch.

Support of TEST DBLOG

When multi-fetch is used, real database calls are only submit to get a new set of records. However, if somebody likes to debug his program with the TEST DBLOG facility, he would neither be able to look at the records processed by his program nor to SNAP at a specific position, because they are filled internally from the multi-fetch buffer.

In order to improve this situation, the multi-fetch handler also triggers calls to TEST DBLOG for every record moved from the multi-fetch buffer. To make these special database calls visible (in the TEST DBLOG list), the command Option1 is set to "<".

Example: TEST DBLOG list break-out

No	Cmd	DB	FNR	Rsp	ISN	ISQ	CID	CID (Hex)	OP	Pgm	Line
2	S1	177	4		89	6	? ??	04000101		TEST006A	0400
3	L1	177	4		108	6	? ??	04000101	MN	TEST006A	0400
4	L1	177	4		299	6	? ??	04000101	<N	TEST006A	0400
5	L1	177	4		418	6	? ??	04000101	<N	TEST006A	0400

Where **No** represents the following:

2 is an ordinary database call without any multi-fetching;

3 is a "real" database call that reads a set of records via multi-fetch (see "M" in column OP) and returns the first record back to the program. The data displayed in the Record Buffer and ISN Buffer do not correspond to the statement in the program, because they contain the values (especially for the ISN-Buffer) of all the records being fetched by the database. The data returned to the program is just the first record, the remaining data will be stored in the multi-fetch buffer.

4-5 are "no real" database calls, but only entries that document that the program has received these records from the multi-fetch buffer (see "<" in column *OP*). All the data in the *CB*, *FB*, *RB*, *SB*, *VB* and *IB* are exactly the same, like when they were fetched from the database.