

Database Update - Transaction Processing

This document describes how Natural performs database updating operations based on transactions.

The following topics are covered:

- Logical Transaction
 - Example of STORE Statement
 - Record Hold Logic
 - Example of GET Statement
 - Backing Out a Transaction
 - Restarting a Transaction
 - Example of Using Transaction Data to Restart a Transaction
-

Logical Transaction

Natural performs database updating operations based on transactions, which means that all database update requests are processed in logical transaction units. A logical transaction is the smallest unit of work (as defined by you) which must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may consist of one or more update statements (DELETE, STORE, UPDATE) involving one or more database files. A logical transaction may also span multiple Natural programs.

A logical transaction begins when a record is put on "hold"; Natural does this automatically when the record is read for updating, for example, if a FIND loop contains an UPDATE or DELETE statement.

The end of a logical transaction is determined by an END TRANSACTION statement in the program. This statement ensures that all updates within the transaction have been successfully applied, and releases all records that were put on "hold" during the transaction.

Example:

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
FIND MYVIEW WITH NAME = 'SMITH'
  DELETE
  END TRANSACTION
END-FIND
END
```

Each record selected would be put on "hold", deleted, and then - when the END TRANSACTION statement is executed - released from "hold".

Note:

The Natural profile parameter OPRB, as set by the Natural administrator, determines whether or not Natural will generate an END TRANSACTION statement at the end of each Natural program. Ask your Natural administrator for details.

Example of STORE Statement

See the following example program in library SYSEXPG:

- STOREX01

Record Hold Logic

If Natural is used with Adabas, any record which is to be updated will be placed in "hold" status until an END TRANSACTION or BACKOUT TRANSACTION statement is issued or the transaction time limit is exceeded.

When a record is placed in "hold" status for one user, the record is not available for update by another user. Another user who wishes to update the same record will be placed in "wait" status until the record is released from "hold" when the first user ends or backs out his/her transaction.

To prevent users from being placed in wait status, the session parameter WH (Wait Hold) can be used (see the Natural Parameter Reference documentation).

When you use update logic in a program, you should consider the following:

- The maximum time that a record can be in hold status is determined by the Adabas transaction time limit (Adabas parameter TT). If this time limit is exceeded, you will receive an error message and all database modifications done since the last END TRANSACTION will be made undone.
- The number of records on hold and the transaction time limit are affected by the size of a transaction, that is, by the placement of the END TRANSACTION statement in the program. Restart facilities should be considered when deciding where to issue an END TRANSACTION. For example, if a majority of records being processed are **not** to be updated, the GET statement is an efficient way of controlling the "holding" of records. This avoids issuing multiple END TRANSACTION statements and reduces the number of ISNs on hold. When you process large files, you should bear in mind that the GET statement requires an additional Adabas call. An example of a GET statement is shown below.

Example of GET Statement

```

DEFINE DATA LOCAL
  1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1)
END-DEFINE
RD. READ EMPLOY-VIEW BY NAME
  IF SALARY (1) > 30000
    GE. GET EMPLOY-VIEW *ISN (RD.)
      compute SALARY (1) = SALARY (1) * 1.15
      UPDATE (GE.)
      END TRANSACTION
  END-IF
END-READ
END

```

On mainframe computers, the placing of records in "hold" status is also controlled by the profile parameter RI, as set by the Natural administrator.

Backing Out a Transaction

During an active logical transaction, that is, before the END TRANSACTION statement is issued, you can cancel the transaction by using a BACKOUT TRANSACTION statement. The execution of this statement removes all updates that have been applied (including all records that have been added or deleted) and releases all records held by the transaction.

Restarting a Transaction

With the END TRANSACTION statement, you can also store transaction-related information. If processing of the transaction terminates abnormally, you can read this information with a GET TRANSACTION DATA statement to ascertain where to resume processing when you restart the transaction.

Example of Using Transaction Data to Restart a Transaction

The following program updates the EMPLOYEES and VEHICLES files. After a restart operation, the user is informed of the last EMPLOYEES record successfully processed. The user can resume processing from that EMPLOYEES record. It would also be possible to set up the restart transaction message to include the last VEHICLES record successfully updated before the restart operation.

```

** Example Program 'GETTRX01'
DEFINE DATA LOCAL
01 PERSON VIEW OF EMPLOYEES
  02 PERSONNEL-ID      (A8)
  02 NAME              (A20)
  02 FIRST-NAME        (A20)
  02 MIDDLE-I          (A1)
  02 CITY              (A20)
01 AUTO VIEW OF VEHICLES
  02 PERSONNEL-ID      (A8)
  02 MAKE              (A20)
  02 MODEL             (A20)
01 ET-DATA
  02 #APPL-ID          (A8) INIT <' '>
  02 #USER-ID          (A8)
  02 #PROGRAM          (A8)
  02 #DATE             (A10)
  02 #TIME             (A8)
  02 #PERSONNEL-NUMBER (A8)
END-DEFINE
*
GET TRANSACTION DATA #APPL-ID #USER-ID #PROGRAM
                    #DATE      #TIME      #PERSONNEL-NUMBER
*
IF #APPL-ID NOT = 'NORMAL'      /* IF LAST EXECUTION ENDED ABNORMALLY
  AND #APPL-ID NOT = ' '
  INPUT (AD=OIL)
  // 20T '*** LAST SUCCESSFUL TRANSACTION ***' (I)
  / 20T '*****'
  /// 25T      'APPLICATION:' #APPL-ID
  / 32T      'USER:' #USER-ID
  / 29T      'PROGRAM:' #PROGRAM
  / 24T      'COMPLETED ON:' #DATE 'AT' #TIME
  / 20T 'PERSONNEL NUMBER:' #PERSONNEL-NUMBER
END-IF
REPEAT
  INPUT (AD=MIL) // 20T 'ENTER PERSONNEL NUMBER:' #PERSONNEL-NUMBER
  IF #PERSONNEL-NUMBER = 99999999

```

```

    ESCAPE bottom
END-IF
FIND1. FIND PERSON WITH PERSONNEL-ID = #PERSONNEL-NUMBER
IF NO RECORDS FOUND
    REINPUT 'SPECIFIED NUMBER DOES NOT EXIST; ENTER ANOTHER ONE.'
END-NOREC
FIND2. FIND AUTO WITH PERSONNEL-ID = #PERSONNEL-NUMBER
IF NO RECORDS FOUND
    WRITE 'PERSON DOES NOT OWN ANY CARS'
END-NOREC
IF *COUNTER (FIND1.) = 1 /* FIRST PASS THROUGH THE LOOP
    INPUT (AD=M)
        / 20T 'EMPLOYEES/AUTOMOBILE DETAILS' (I)
        / 20T '-----'
    /// 20T 'NUMBER:' PERSONNEL-ID (AD=O)
        / 22T 'NAME:' NAME ' ' FIRST-NAME ' ' MIDDLE-I
        / 22T 'CITY:' CITY
        / 22T 'MAKE:' MAKE
        / 21T 'MODEL:' MODEL
    UPDATE (FIND1.) /* UPDATE THE EMPLOYEES FILE
ELSE /* SUBSEQUENT PASSES THROUGH THE LOOP
    INPUT NO ERASE (AD=M) ////////// 20T MAKE / 20T MODEL
END-IF
    UPDATE (FIND2.) /* UPDATE THE VEHICLES FILE
MOVE *APPLIC-ID TO #APPL-ID
MOVE *INIT-USER TO #USER-ID
MOVE *PROGRAM TO #PROGRAM
MOVE *DAT4E TO #DATE
MOVE *TIME TO #TIME
END TRANSACTION #APPL-ID #USER-ID #PROGRAM
                #DATE #TIME #PERSONNEL-NUMBER
END-FIND /* FOR VEHICLES (FIND2.)
END-FIND /* FOR EMPLOYEES (FIND1.)
END-REPEAT /* FOR REPEAT
STOP /* Simulate abnormal transaction end
END TRANSACTION 'NORMAL '
END

```