Using ADL Files with Natural/Adabas

This chapter covers the following topics:

- Introduction
- Consistency Interface
- Restrictions when using Natural/Adabas
- Improve the Natural Access to Migrated Files
- Error Situations and Consistency Response Codes
- Availability of the Consistency Interface
- Example Programs

Introduction

After a DL/I data base was converted to one or more ADL files, these files can be accessed and manipulated with Natural applications or other programs using direct Adabas calls. This section explains:

- how you can do this,
- how ADL guarantees the integrity of the data as expected by original DL/I applications,
- which restrictions apply for updates from Natural/Adabas applications.
- how to improve the access to the migrated file for Natural applications.

Note:
Set the Natural parameter `ADAMODE` to "0" when you run Natural programs against the ADL Consistency.

Consistency Interface

When hierarchical structures, like those being defined in DL/I data bases, are mirrored into a table like structure as those of ADL files, the original structure of the data has to be preserved by either the use of physical pointers or foreign keys. The latter may be considered as logical pointers. ADL maintains both kinds of pointers automatically.

The foreign keys maintained by ADL for a given segment type are built up in a similar way as a DL/I concatenated key. In other words, the foreign key stored together with a segment occurrence determines its position in the data base by identifying its parent segments (if there are any). ADL introduces the term "partial concatenated key" (PCK) for the foreign key of *one* particular segment type.

The physical pointers are needed for DL/I applications. Whenever a DL/I application issues a data base call, this call is intercepted by the ADL CALLDLI Interface. Implicitly, a DL/I data base call is serviced by referring to the physical pointers. We say implicitly, because these physical pointers are determined by the actual position of the segment occurrences in the data base. Based on this, the ADL CALLDLI Interface then maintains the foreign keys automatically, though they are never used directly by DL/I applications.

In case of Natural/Adabas applications, the situation is just the other way around. An Adabas call on ADL files supplies the segment data and all PCKs of its parent segments. In order for DL/I applications to be able to "see" these data as well, the corresponding physical pointers have to be built up. This task is performed by the ADL Consistency Interface.

The ADL Consistency Interface intercepts Adabas calls. On the first level, all kind of retrieval calls (like Adabas L3, S1 etc.) are routed directly to Adabas. On a second level, only calls referring to ADL files will be considered. If necessary, the Consistency automatically constructs the physical pointers for the record to be inserted, updated or deleted based on the PCKs provided by the user.

In a later stage of the data base conversion process, you might want to phase out the old DL/I applications completely. Therefore, the further operation of the ADL Consistency Interface may become obsolete. For this reason, the Natural applications or programs using direct Adabas calls should originally be designed as if the Consistency Interface is not existent.

Note that the installation and operation of the Consistency Interface is described earlier in this documentation.

Summary

- the CALLDLI Interface automatically sets and maintains the PCKs of all parent segments,
- the Consistency Interface sets and maintains the physical pointers needed to preserve the hierarchical structure of the data base,
- the Consistency Interface is necessary to update ADL files with Natural or programs using direct Adabas calls, when access to these files for DL/I applications must still be provided.

Restrictions when using Natural/Adabas

Natural applications or programs using Adabas direct calls follow exactly the same syntax in accessing either a normal Adabas file or an ADL file. The only difference is, that the applications may receive response codes from the ADL Consistency Interface, if it is active.

The rules having to be obeyed when manipulating ADL files with Natural/Adabas programs are derived directly from the fact that the hierarchical structure of the data, as seen from DL/I applications, has to preserved. The ADL Consistency Interface is a product which guarantees the referential integrity of your data.

The following is an example of the hierarchical structure of a DL/I data base consisting of four segments types:

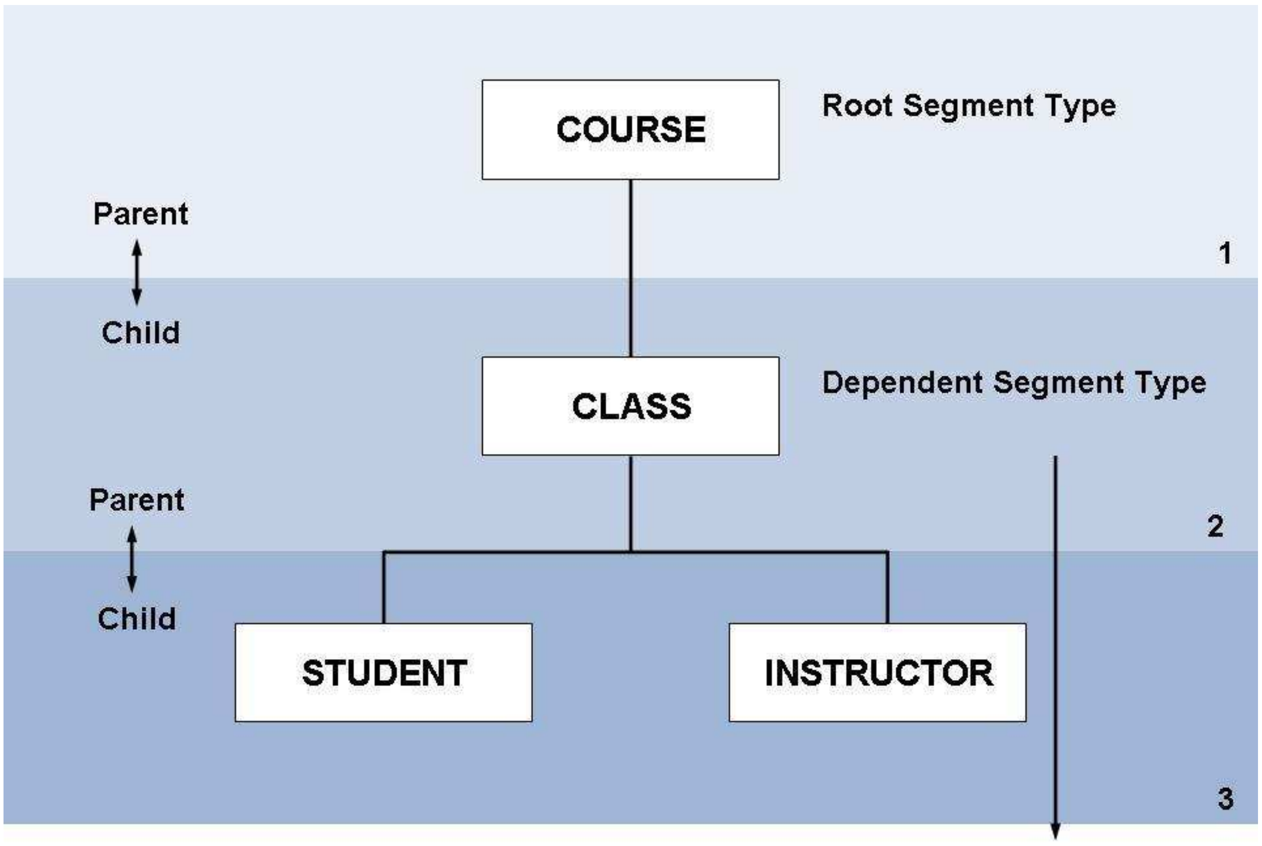

Figure: Sample data base used in our examples

In DL/I, the segments STUDENT and INSTRUCTOR are child segments to the parent segment CLASS, which in turn is a child segment to COURSE. The parent segments are identified by a course number or class number respectively.

Transformed to an ADL file the course number is denoted the PCK of the record COURSE and serves as a link for the records COURSE and CLASS. ADL stores the course number together with the CLASS data as shown in the following table:

DL/I Segment Type	ADL Record Type
COURSE	COURSE
CLASS	PCK-COURSE, CLASS
STUDENT	PCK-COURSE, PCK-CLASS, STUDENT
INSTRUCTOR	PCK-COURSE, PCK-CLASS, INSTRUCTOR

Referential integrity in this context means that each STUDENT/INSTRUCTOR record occurrence must refer to an existing CLASS record occurrence, which in turn must also refer to an existing COURSE record occurrence.

In order to allow the Consistency Interface to maintain the referential integrity, relation data have to be supplied with each update request to an ADL file. For insert and replace calls (Adabas N1 or A1) this simply means that you have to supply the foreign keys of all related record types. In DL/I calls this means supplying the concatenated key. Furthermore, when replacing a record, you must not alter the foreign keys of its parent segment.

Delete calls may not attempt to delete parent segment occurrences which have dependent child segment occurrences. For ADL files this means, that record occurrences without PCKs referred to by other record occurrences must be deleted first. Considering the above example, note that a record occurrence of CLASS can only be deleted after all record occurrences of STUDENT and INSTRUCTOR have been deleted.

Summary

- when inserting records in an ADL file, specify the corresponding foreign keys or PCKs for the related record type;
- when replacing records in an ADL file, specify the corresponding foreign keys or PCKs for the related record type but NEVER alter the PCKs;
- when deleting a record occurrence, be sure that it is not referred to by PCK fields in other record occurrences;
- the Consistency Interface will preserve the referential integrity of the data base.

Improve the Natural Access to Migrated Files

When ADL migrates the DL/I structure, the generated Adabas file layout is optimized to satisfy the needs of the DL/I applications. Therefore only the root sequence field, the physical pointer fields ("Z1-field") and the secondary index fields are defined as descriptors. From these fields only the root sequence field should be used by Natural applications because the other fields are ADL internal fields.

Natural applications read the hierarchy through the PCK fields. If the application should access the data of a dependent segment, it is recommended to create an Adabas superdescriptor build up by the PCK fields of the parent segments. If the segment itself has a sequence field, the corresponding Adabas field should be added to the superdescriptor as well.

If Natural applications should access data in a secondary index like manner, any other field (like a sequence field of a dependent segment) can be defined as descriptor. If you build up new superdescriptors take care that only those fields are included which belong to the same segment plus PCK fields of parents of the segment. If you use fields from different segments, one part will always be empty and the superdescriptor will not give you any value.

It is recommended that you define only those descriptors and superdescriptors which are really used by any application. Every descriptor cost some performance not only in the Natural applications but also in the DL/I applications and when utilities run. On the other hand if you have use for a descriptor or superdescriptor, you should define it. A non-descriptor search gives you the poorest performance.

Error Situations and Consistency Response Codes

Whenever you violate the rules for manipulating ADL files defined earlier in this section, the ADL Consistency Interface will return a non-zero response code in the Adabas control block. For Natural applications, this response code is available in the system variable *ERROR-NR. The Adabas response code returned is "216" and the corresponding Natural *ERROR-NR is "3216".

A more detailed error message can then be obtained from the ADL Consistency Interface by submitting an Adabas S1 call conforming to a defined standard. This is in detail explained in the documentation *ADL Messages and Codes*. Natural programmers may simply obtain this error message by a call to the ADL supplied Natural subprogram ADLERROR, as in the example below:

```
IF *ERROR-NR = 3216
  CALLNAT 'ADLERROR' #ERRMESS
END-IF
```

providing an alphanumeric variable "#ERRMESS" of at least 80 bytes length.

The error text returned has the following layout:

```
'ADLxxxx - error message  .... '
```

where xxxx is a four character error code. For an explanation of the error code, please refer to the documentation *ADL Messages and Codes*.

As explained above, the ADL Consistency Interface will guarantee the referential integrity of your ADL files. However, we recommend a programming style, which avoids updates resulting in ADL Consistency Interface response codes. In other words, the logic to preserve the referential integrity of the data should as far as possible be included in the application program itself.

Example

Consider our example data base: Before placing a delete call for a record occurrence of CLASS, place a read call to make sure that no record occurrences of STUDENT/ INSTRUCTOR still exist.

The reason for our recommendation is, that in a later stage of the data base conversion process the ADL Consistency Interface could become obsolete. Then, the logic of the application programs might be affected by not receiving any Consistency response codes.

Availability of the Consistency Interface

Natural programs may check the availability of the ADL Consistency Interface by calling the ADL supplied Natural subprogram ADLACTIV. It returns a 2-bytes integer value. If the ADL Consistency Interface is active the value '0', otherwise the value '8' is returned. Under CICS, if the ADL user exit ADLEXITB is installed but the ADL Interfaces are not activated, a value '4' is returned.

In general, a program which updates migrated data, should only run when the response from ADLACTIV is '0'.

Example

```
DEFINE DATA LOCAL
1 #RSP(I2)
END-DEFINE
CALLNAT ’ADLACTIV’ #RSP
DECIDE ON FIRST VALUE OF #RSP
VALUE 0
  WRITE ’ADL Consistency is active.’
VALUE 4
  WRITE ’ADL user exit installed, but ADL is not active.’
VALUE 8
  WRITE ’ADL Consistency is not active.’
NONE
  WRITE ’Unexpected response from ADL Consistency.’
END-DECIDE
END
```

Example Programs

The following Natural programs demonstrate how to code with respect to the referential integrity. The full sources can be found on the SYSADLIV Natural library.

Program INS-STUD: Insert a Student

Before a dependent segment is inserted, it should be verified that the parents, i.e. the path to the segment, exists.

```
* Check if the path to the student exits
*
  FIND (1) COURSEDB-CLASS WITH CLASSNO = #CLASSNO
      WHERE PCK-COURSENO = #COURSENO
    IF NO RECORDS FOUND
      REINPUT ’COURSE / CLASS NOT FOUND !’
    END-NOREC
*
* Store the data in the data base
*
    MOVE #COURSENO TO COURSEDB-STUDENT-UPD.PCK-COURSENO
    MOVE #CLASSNO TO COURSEDB-STUDENT-UPD.PCK-CLASSNO
    MOVE #SURNAME TO COURSEDB-STUDENT-UPD.SURNAME
    MOVE #FILLER-AB TO COURSEDB-STUDENT-UPD.FILLER-AB
    STORE COURSEDB-STUDENT-UPD
    END OF TRANSACTION
  END-FIND
```

Program DEL-COUR: Delete a Course

Before a parent is deleted, it should be verified that it has no dependents. The following program does not perform the delete if there are dependents

```
* Check if the data exits
*
COURSE. FIND (1) COURSEDB-COURSE WITH COURSENO = #COURSENO
    IF NO RECORDS FOUND
      REINPUT ’COURSE NOT FOUND !’
    END-NOREC
*
* Check if there are dependents
```

```
*
    RESET #NBR
CLASS. READ COURSEDB-CLASS WHERE PCK-COURSENO = #COURSENO
      MOVE *COUNTER TO #NBR
      WRITE #NBR '. CLASS FOUND UNDER THIS COURSE:' CLASSNO CLASSNAME
    END-READ
*
    IF #NBR = 0  /* NO DEPENDENTS
*
* Delete the data
*
      DELETE (COURSE.)
      END OF TRANSACTION
      WRITE 'COURSE DELETED!'
    END-IF
  END-FIND
```

Program CDELCOUR: Cascaded delete of a Course

When DL/I deletes a segment, all dependents are deleted automatically. This is named “*cascaded delete*”. The program CDELCOUR (together with the programs called by it) is an example on how to code a cascaded deletion with Natural.

```
FIND (1) COURSEDB-COURSE WITH COURSENO = #COURSENO
    IF NO RECORDS FOUND
      MOVE 'Course not found!'
        TO #ADL-MESS
      ESCAPE BOTTOM
    END-NOREC
*   Delete dependents recursively
    CALLNAT 'CASDELCL' #COURSENO #N-CL #N-ST #N-IP #RESPONSE
    IF #RESPONSE NE 0
      ESCAPE BOTTOM
    END-IF
*   All dependents are deleted -> delete the COURSE
    DELETE
    MOVE 1 TO #N-SUM
  END-FIND
```