Fast Path Database

Information Management software

Fast Path Database

- Fast Path 64-Bit Buffer Manager Enhancements
- Miscellaneous Enhancements
- Fast Path Secondary Indexing Support

Fast Path 64-Bit Buffer Manager Enhancements

.

Fast Path 64-Bit Buffer Manager Enhancements

- IMS 11 introduced optional Fast Path 64-bit Buffer Manager
  - Database buffers above the bar (in 64-bit storage)
  - Multiple subpools with different buffer sizes
  - IMS automatically determined size and number of buffers
  - Buffer pools dynamically expanded
  - Defined in DFSDFxxx PROCLIB member

```
<SECTION=FASTPATH>
FPBP64=Y,FPBP64M=xxxxxxxx
```

    - FPBP64=Y invoked 64-bit buffer manager
    - FPBP64M= set maximum storage used

672

This is a review of the Fast Path 64-bit buffer manager support added to IMS in IMS 11. It provided for database buffers above the 2GB bar, that is, in 64-bit virtual storage. The 64-bit buffer manager created multiple subpools with different buffer sizes. This was advantageous for users with area data sets of different CI sizes. The 64-bit buffer manager created an initial allocation of subpools based on the number of areas of each CI size. It automatically created more buffers in a subpool when they were needed. The 64-bit buffer manager was used when FPBP64=Y was specified in the FASTPATH section of the DFSDFxxx PROCLIB member. The maximum amount of 64-bit storage used could be limited with the FPBP64M= parameter.

Fast Path 64-Bit Buffer Manager Enhancements

- IMS 12 enhancements to the Fast Path 64-bit Buffer Manager
  - User control over initial buffer pool storage
  - Dynamic pre-extension and compression of buffer pools
  - Additional buffers moved from ECSA to 64-bit storage
  - Enhanced QUERY POOL TYPE(FPBP64) command output
- Benefits
  - More user controls
  - Improved management of 64-bit buffers
  - Enhanced use of 64-bit storage

IMS 12 enhances the Fast Path 64-bit buffer manager which was introduced in IMS 11.

IMS 12 allows the user to specify the initial amount of 64-bit storage used for the buffer pool. Buffer pools are pre-expanded, that is, expanded in anticipation of future needs. They are compressed when the use of a subpool drops. IMS 12 moves some buffers that were still in ECSA to 64-bit storage. Finally, IMS 12 enhances the QUERY POOL TYPE(FPBP64) command output.

User Control of Initial Buffer Storage

- New parameter to set total buffer pools to 25% of DBBF specification
  - Default is FPBP64D=N
    - Initial allocation about 1M
    - Number of buffers of each size is determined by IMS based on the number of areas of each CI size
  - FPBP64D=Y
    - Initial allocation is 25% of DBBF specification, distributed among subpools based on the number of areas of each CI size
    - Example:
      - DBBF=8000; 100 1K CI areas; 200 2K CI areas; 700 4K CI areas
      - Results:
        - 200 1K buffers
        - 400 2K buffers
        - 1400 4K buffers

100+200+700 = 1000 areas; 25% of 8000 = 2000 buffers
1K areas: 100 of 1000 areas is 10%; 10% of 2000 = 200
2K areas: 200 of 1000 areas is 20%; 20% of 2000 = 400
4K areas: 700 of 1000 areas is 70%; 70% of 2000 = 1400

IMS 12 allows the user to control the amount of storage used for the initial allocation of database buffers in 64-bit storage. If FPBP64D=Y is specified in the FASTPATH section of the DFSDFxxx PROCLIB member, the DBBF= execution parameter is used to determine the initial number of buffers. FPBP64D=N is the default. This default leaves the initial allocation as it was in IMS 11. If FPBP64D=Y is specified but there is no DBBF specification, the initial allocation is as it was in IMS 11. If FPBP64D=Y and a value is specified for DBBF, the initial number of buffers is twenty-five percent of the DBBF specification. These buffers are allocated among different buffer sizes based on the number of areas with different CI sizes. In the example shown here, the DBBF specification is 8000. This means that 2000 (25% of 8000) buffers will be used for the initial allocation. In the example, there are 100 areas with a 1K CI size, 200 areas with a 2K CI size, and 700 areas with a 4K CI size. Since 10% of the areas have a 1K CI size, 200 (10% of 2000) 1K buffers are initially allocated. Similarly, 400 (20% of 2000) 2K buffers are initially allocated and 1400 (70% of 2000) 4K buffers are initially allocated.

Subpool Pre-extension

- Subpools are expanded before buffers are required
  - Avoids wait-for-buffer conditions
    - IMS 11 expanded subpools only when a wait for a buffer occurred
    - IMS 12 expands a subpool when it is almost out of buffers
  - Specified with FPBP64E=Y (default)
  - Disabled with FPBP64E=N

IMS 11 expands a 64-bit subpool when a DL/I call requires a buffer but none is available. IMS 12 has an option to expand subpools in anticipation of the need for more buffers. This option is the default. It may be turned off by specifying FPBP64E=N in the DFSDFxxx PROCLIB member. The expansion of subpools before new buffers are required avoids application waits for the creation of new buffers.

When a subpool is almost out of available buffers, the extention process is initiated asynchronously. As the volume of buffer requests increase, the subpool extension process will increase the pace at which subpools are extended. By the time the additional buffers are required, the subpool should have been extended, avoiding wait-for-buffer conditions.

Subpool Compression and Deletion

- Subpools are compressed when buffers are unused
  - Frees unused buffers in the subpool
    - IMS 11 did not compress pools
  - Specified with FPBP64C=Y (default)
  - Disabled with FPBP64C=N
- Subpools will be deleted when not used for 24 hours
  - Subpools may be recreated if CIs of buffer size are used later
  - Specified and disabled with FPBP64C= parameter

IMS 12 also adds the capability to compress subpools when there are substantial unused buffers. Subpools are compressed by reducing the number of buffers in the subpool. This capability is the default. It may be disabled by specifying FPBP64C=N.

Subpools may also be deleted. That is, all of the buffers of a certain size may be deleted. This will only be done when none of the buffers of this size have been used for over 24 hours. If buffers of this size are needed again, the subpool will be rebuilt.

The compression and deletion actions are the default. They may be disabled by specifying FPBP64C=N in the FASTPATH section of the DFSDFxxx PROCLIB member.

Buffers Moved from ECSA to 64-bit Storage

- IMS 12 uses 64-bit buffers for FLD calls
  - ECSA buffers were used by IMS 11
- Emergency restart uses 64-bit buffers for SDEPs
  - ECSA buffers were used by IMS 11
  - May be turned off with FPBP64SR=N

677

The use of the 64-bit Fast Path buffer manager in IMS 11 did not put all buffers in 64-bit storage. Buffers in ECSA were used for MSDBs, SDEP inserts and FLD calls by IMS 11. Additionally, ECSA storage was used for buffer headers and control blocks.

IMS 11 emergency restart uses ECSA buffers for all SDEP processing.

IMS 12 uses 64-bit buffers for FLD calls.

IMS 12 emergency restart uses 64-bit buffers for SDEPs unless FPBP64SR=N is specified.

Enhanced QUERY POOL TYPE(FPBP64) Command

- QUERY POOL TYPE(FPBP64) is enhanced
  - SHOW(STATISTICS) added
    - Shows a subset of the data returned by SHOW(ALL)
  - SHOW(ALL) output is enhanced
    - Shows extended private (EPVT) use
    - Shows status of subpools
      - e.g. being compressed, deleted, etc.
    - Reformatted for ease of use

678

IMS 11 provided the QUERY POOL TYPE(FPBP64) command. It displayed information on the use of buffers by the 64-bit Fast Path buffer manager.

IMS 12 enhances the command. It provides the SHOW(STATISTICS) option which provides a subset of the data returned with SHOW(ALL) which was the only SHOW option available in IMS 11. SHOW(ALL) provided so much information that it could be difficult to find the most interesting data.

The QUERY command in IMS 12 also provides data on extended private (EPVT) use. It provides new status information for subpools. The status shows if a pool is being compressed, expanded, or deleted.

QUERY POOL TYPE(FPBP64) SHOW(STATISTICS)

Response for: QUERY POOL TYPE(FPBP64) SHOW(STATISTICS) More: >

Buf_Size	MbrName	CC	SPT	Tot_Buf	Buf_Use	Buf_Avl	%Use	HWM	EPVT_Tot	ECSA_Tot	64b_Tot
Total				200	8	192	4	168	2K	282K	244K
512	SYS3	0	C	32	0	32	0	146	780	72K	16K
1024	SYS3	0	C	36	0	36	0	0	156	16K	36K
2048	SYS3	0	C	16	0	16	0	0	156	7K	32K
4096	SYS3	0	C	40	0	40	0	0	156	18K	160K
512	SYS3	0	S	32	8	24	25	14	156	30K	0
1024	SYS3	0	S	28	0	28	0	0	156	40K	0
2048	SYS3	0	S	8	0	8	0	0	156	20K	0
4096	SYS3	0	S	8	0	8	0	8	312	71K	0

C – 64-bit
S – ECSA

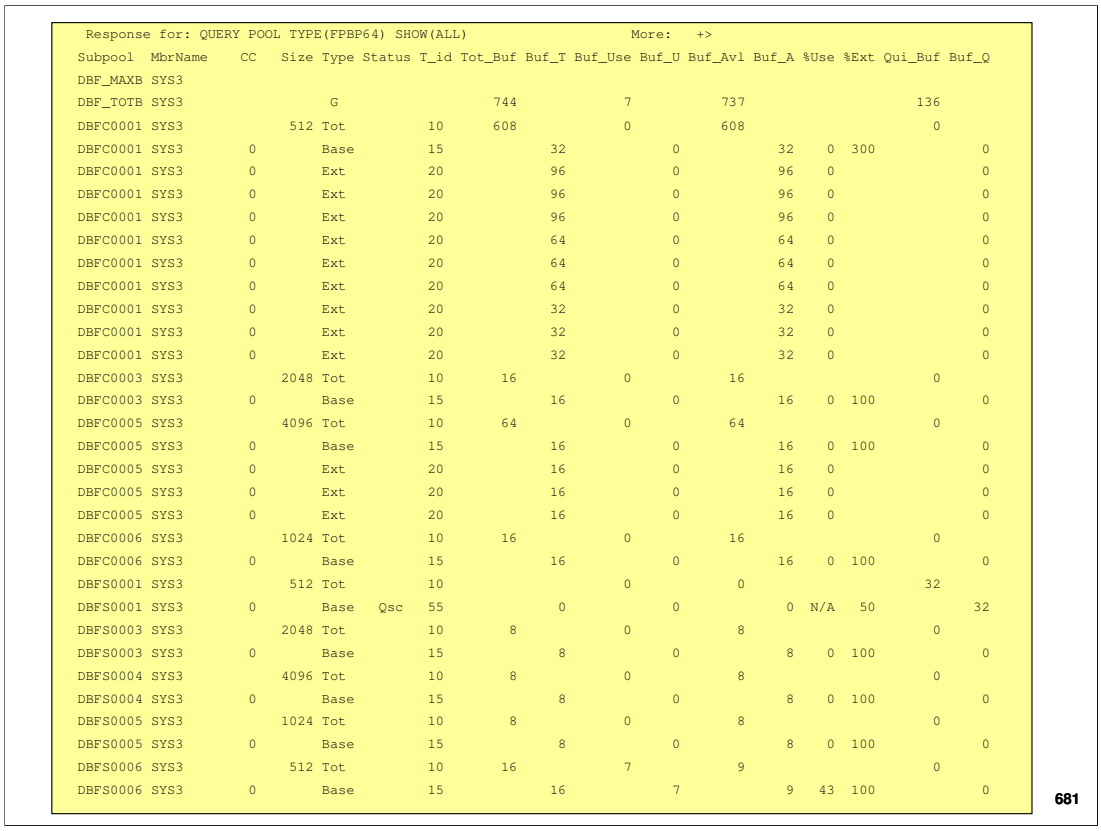
679

This is an example of the output of a QUERY POOL TYPE(FPBP64) command with the SHOW(STATISTICS) option. There is one line for the total, one line for each pool in 64-bit storage and one line for each pool in ECSA. The C or S in the SPT column indicates if the pool is in 64-bit (C) or ECSA (S) storage. Total buffers, buffers currently in use, buffers available, the per cent in use, and the “high water mark” or maximum number of buffers used are shown in following columns. The last three columns show the storage used for the buffer pools. 64-bit buffer pools use storage in extended private and ECSA for control blocks.

QUERY POOL TYPE(FPBP64) SHOW(ALL)

- The following two pages show an example of the output of a QUERY POOL TYPE(FPBP64) SHOW(ALL) command
  - The first page shows the first screen returned using the TSO SPOC
  - The second page shows the results of scrolling to the right

This next two pages show an example of the output of a QUERY POOL TYPE(FPBP64) command with the SHOW(ALL) option. The second page is the result of scrolling to the right.



```
 Response for: QUERY POOL TYPE(FPBP64) SHOW(ALL)                  More:   +>
Subpool  MbrName    CC   Size Type Status T_id Tot_Buf Buf_T Buf_Use Buf_U Buf_Avl Buf_A %Use %Ext Qui_Buf Buf_Q
DBF_MAXB SYS3
DBF_TOTB SYS3                G                  744            7           737                    136
DBFC0001 SYS3            512 Tot          10      608            0           608                      0
DBFC0001 SYS3        0       Base         15             32            0            32    0  300             0
DBFC0001 SYS3        0       Ext          20             96            0            96    0                  0
DBFC0001 SYS3        0       Ext          20             96            0            96    0                  0
DBFC0001 SYS3        0       Ext          20             96            0            96    0                  0
DBFC0001 SYS3        0       Ext          20             64            0            64    0                  0
DBFC0001 SYS3        0       Ext          20             64            0            64    0                  0
DBFC0001 SYS3        0       Ext          20             64            0            64    0                  0
DBFC0001 SYS3        0       Ext          20             32            0            32    0                  0
DBFC0001 SYS3        0       Ext          20             32            0            32    0                  0
DBFC0001 SYS3        0       Ext          20             32            0            32    0                  0
DBFC0003 SYS3           2048 Tot          10       16            0            16                      0
DBFC0003 SYS3        0       Base         15             16            0            16    0  100             0
DBFC0005 SYS3           4096 Tot          10       64            0            64                      0
DBFC0005 SYS3        0       Base         15             16            0            16    0  100             0
DBFC0005 SYS3        0       Ext          20             16            0            16    0                  0
DBFC0005 SYS3        0       Ext          20             16            0            16    0                  0
DBFC0005 SYS3        0       Ext          20             16            0            16    0                  0
DBFC0006 SYS3           1024 Tot          10       16            0            16                      0
DBFC0006 SYS3        0       Base         15             16            0            16    0  100             0
DBFS0001 SYS3            512 Tot          10                     0             0                     32
DBFS0001 SYS3        0       Base   Qsc   55              0            0             0  N/A   50            32
DBFS0003 SYS3           2048 Tot          10        8            0             8                      0
DBFS0003 SYS3        0       Base         15              8            0             8    0  100             0
DBFS0004 SYS3           4096 Tot          10        8            0             8                      0
DBFS0004 SYS3        0       Base         15              8            0             8    0  100             0
DBFS0005 SYS3           1024 Tot          10        8            0             8                      0
DBFS0005 SYS3        0       Base         15              8            0             8    0  100             0
DBFS0006 SYS3            512 Tot          10       16            7             9                      0
DBFS0006 SYS3        0       Base         15             16            7             9   43  100             0
```

681

This is the first screen from the response to the QUERY POOL TYPE(FPBP64) SHOW(ALL) command. The first line after the header has no information for the columns shown on this screen. The second line shows the totals for all pools. Following lines show the total for each pool followed by data for each extent of a pool. These are the base extent and each extension of the pool.

The columns and their meanings are:

Subpool: The name of the subpool. This is the internal name of the pool where xxxx is a numeric value.

DBFCxxxx A common subpool used for DEDB data. The buffers reside in 64-bit addressable storage.

DBFSxxxx A system subpool used for all other buffer requests, including IMS internal buffers. The buffers reside in ECSA.

MbrName: The IMS identifier. The same as in other type-2 command responses.

CC: Command completion code. The same as in other type-2 command responses.

Size: The buffer size

Type: Describes what this row describes

G: This row contains overall totals for the entire buffer pool.

Tot: The total values for the subpool and extents with the name of the subpool in the SUBPOOL column.

Base: This is the base section of the subpool. It does not include the extent values.

Ext: This is an extent for the subpool. It does not include the base section of the subpool.

Status: The status of this base or extent. QSC-quiescing (no buffers in use); QSCW-waiting for quiesce; Del-being deleted

T_id: Token id associated with the base or extent

Tot_Buf: The total number of buffers in this subpool; Buf_T: Buffers in this base section or extent

Buf_Use or Buf_U: The number of buffers being used

Buf_Ava or Buf_A: The number of buffers available for use

Qui_Buf or Buf_Q: The number of buffers quiesced (for deletion of extent)

```
  Response for: QUERY POOL TYPE(FPBP64) SHOW(ALL)
Subpool    Size Type  EPVT_Tot EPVT_T ECSA_Tot ECSA_Buf ECSA_B ECSA_Oth ECSA_O 64b_Tot 64b_Buf TimeCreate
DBF_MAXB                                                                          2048M
DBF_TOTB        G          4K            510K     108K            397K           804K
DBFC0001    512 Tot         2K                                     266K           304K
DBFC0001        Base             1K                                          18K             16K 2010.148 15:05:04.95
DBFC0001        Ext            156                                           42K             48K 2010.148 15:25:02.58
DBFC0001        Ext            156                                           42K             48K 2010.148 15:25:02.51
DBFC0001        Ext            156                                           42K             48K 2010.148 15:25:02.41
DBFC0001        Ext            156                                           28K             32K 2010.148 15:25:02.36
DBFC0001        Ext            156                                           28K             32K 2010.148 15:24:55.49
DBFC0001        Ext            156                                           28K             32K 2010.148 15:17:10.70
DBFC0001        Ext            156                                           14K             16K 2010.148 15:17:10.29
DBFC0001        Ext            156                                           14K             16K 2010.148 15:17:09.72
DBFC0001        Ext            156                                           14K             16K 2010.148 15:17:09.16
DBFC0003   2048 Tot         156                                            7K             32K
DBFC0003        Base            156                                            7K             32K 2010.148 15:05:04.95
DBFC0005   4096 Tot         936                                           28K            256K
DBFC0005        Base            468                                            8K             64K 2010.148 15:09:05.45
DBFC0005        Ext             156                                            7K             64K 2010.148 15:24:43.27
DBFC0005        Ext             156                                            7K             64K 2010.148 15:24:43.24
DBFC0005        Ext             156                                            7K             64K 2010.148 15:24:43.29
DBFC0006   1024 Tot           0                                             7K             16K
DBFC0006        Base              0                                            7K             16K 2010.148 15:09:05.46
DBFS0001    512 Tot         156                       30K             14K
DBFS0001        Base            156                             30K             14K                 2010.148 15:05:04.95
DBFS0003   2048 Tot         156                       20K              3K
DBFS0003        Base            156                             19K              4K                 2010.148 15:05:04.95
DBFS0004   4096 Tot         156                       36K              3K
DBFS0004        Base            156                             35K              4K                 2010.148 15:09:05.46
DBFS0005   1024 Tot           0                       12K              3K
DBFS0005        Base              0                             11K              4K                 2010.148 15:09:05.46
DBFS0006    512 Tot           0                       15K              7K
DBFS0006        Base              0                             15K              7K                 2010.148 15:09:05.46
```

682

This is the result of scrolling to the right.

The columns and their meanings are:

The meanings of the columns shown here are:
EPVT_Tot or EPVT_T: Extended private storage used by this pool or extent
ECSA_Tot or ECSA_T: ECSA storage used by this pool or extent
ECSA_Buf or ECSA_B: ECSA storage used by this pool or extent for buffers and their associated control blocks
ECSA_Oth or ECSA_O: Other ECSA storage used by this pool or extent
64b_Tot or 64b_Buf: 64-bit buffer storage used by this pool or extent
TimeCreate: The time that the base or extent was created

Fast Path 64-Bit Buffer Manager Enhancements

- Benefits
  - User control of initial buffer allocation
  - Better dynamic management of buffers
  - More buffers moved to 64-bit storage
  - Enhanced QUERY command

683

IMS 12 enhances the Fast Path 64-bit buffer manager which was introduced in IMS 11.

IMS 12 allows users to specify the initial amount of 64-bit storage used for the buffer pool. Buffer pools are pre-expanded, that is, expanded in anticipation of future needs. They are compressed when the use of a subpool drops. They are deleted when not used for over 24 hours. IMS 12 moves some buffers that were still in ECSA to 64-bit storage. Finally, IMS 12 enhances the QUERY POOL TYPE(FPBP64) command output.

Fast Path Miscellaneous Enhancements

Fast Path Miscellaneous Enhancements

- Other IMS 12 enhancements to Fast Path
  - Reduced logging for changed data capture (type x'99' log records)
  - Full segment logging option
  - Improved diagnostic message for data sharing

IMS 12 includes several miscellaneous enhancements to Fast Path. They are explained on the following pages.

Reduced Logging for DEDB Changed Data Capture

- IMS 12 adds option to reduce logging for asynchronous changed data capture
  - Before IMS 12 asynchronous changed data capture writes 'before' and 'after' image log records (x'99')
  - IMS 12 has option not to write these records for DLET calls or 'before' records for REPL calls for DEDBs
  - Specification on EXIT= parameter of DBD and SEGM macros in DBDGEN
    - DLET|NODLET
      - DLET: x'99' record is written for DLET calls (default)
      - NODLET: x'99' record is not written for DLET calls
    - BEFORE|NOBEFORE
      - BEFORE: before image x'99' record is written for REPL calls (default)
      - NOBEFORE: before image x'99' record is not written for REPL calls
- Benefit
  - Reduced logging for asynchronous changed data capture

686

Some users want to use asynchronous changed data capture; however, they do not want to write log records for before images. IMS 12 allows users to specify that these before image log records are not to be written. This is specified with new values on the EXIT= parameter of the DBD or SEGM macro of DBDGEN.

The new values are:

- DLET which writes the before image log record for DLET calls. This is the default. It is also the action taken by previous versions of IMS.
- NODLET which does not write the before image log record for DLET calls.
- BEFORE which writes the before image log record for REPL calls. This is the default. It is also the action taken by previous versions of IMS.
- NOBEFORE which does not write the before image log record for REPL calls.

The benefit of this change is to reduce logging volumes for asynchronous data capture for users who want only after image log records.

Full Segment Logging Option

- IMS 12 provides option to log entire segment for REPL calls of DEDBs
  - ISRT and DLET always log the entire segment
  - Before IMS 12 only changed data in segment was logged for REPL calls
  - Specification in DBRC:

```
INIT.DB DBD(name) NOFULLSEG|FULLSEG …

CHANGE.DB DBD(name) NOFULLSEG|FULLSEG …

INIT.DB DBD(name) AREA(aname) NOFULLSEG|FULLSEG …

CHANGE.DB DBD(name) AREA(aname) NOFULLSEG|FULLSEG …
```

- Benefit
  - Log Archive exit has access to entire segment
    - Less logging than Asynchronous Changed Data Capture

687

IMS 12 provides new options to log entire DEDB segments when a REPL call replaces some of the data in a segment. Previously, only the changed data was logged in the x'5950' log record. The option to log the entire segment is specified for either the DEDB database or the area. There are new keywords for the INIT.DB, CHANGE.DB, INIT.AREA, and CHANGE.AREA DBRC commands. FULLSEG indicates that the entire segment is to be logged. NOFULLSEG indicates that only changed data is logged. The default is to log only the changed data. If FULLSEG is specified on the INIT.DB or CHANGE.DB command without the AREA(aname) specified, the entire segment will be logged for all segments in the database unless NOFULLSEG is specified for an area with either the INIT.DB or CHANGE.DB command with AREA(aname) specified.

The new option is especially useful for users of the Log Archive exit who want access to the entire segment for REPL calls. Without this enhancement, users would need to invoke Asynchronous Changed Data Capture. It writes x'99' log records in addition to the x'5950' log records. The new option provides the data needed without writing additional log records.

Improved Diagnostic Message for DEDB Data Sharing

- New DFS0066I message issued when data sharing system does not respond to notify message
  - Data sharing systems must synchronize some activities
    - /START AREA GLOBAL, UOW lock initialization, etc.
    - Synchronization is done with IRLM notify messages
    - If IMS system fails to respond within time limit, other (waiting) IMS system issues DFS3770W message
      - This message does not identify the system which has failed to respond
  - New DFS0066I message identifies the system which has responded
    - User may need to cancel the IMS which has not issued this message

```
DFS0066I A NOTIFY RESPONSE HAS COME BACK FROM imsname
```

- Benefit
  - User may more easily and quickly resolve the problem

688

IMS 12 includes a new diagnostic message for data sharing. The new DFS0066I message is issued when a data sharing system does not respond to a notify message. Notify messages are used to send messages between data sharing IMS systems. For example, they are sent to synchronize the initialization of the use of UOW locks. These messages are sent by one IMS system to all of its data sharing partners. All partners must respond before the processing may continue. IMS sends a DFS3770W message if all partners do not respond within a time limit. The DFS3770W tells the user that the timeout situation has occurred but does not identify which IMS system has not responded. IMS 12 uses the new DFS0066I message to assist the user. This message is sent by all systems responding to the notify message. If a DFS3770W message is issued, the user can identify the system which has not responded by determining for which system the DFS0066I message has not been sent. All DFS0066I messages are issued by the IMS system which sent the original notify message. This new message makes it easier for users to identify the failing system and to resolve the problem more quickly.

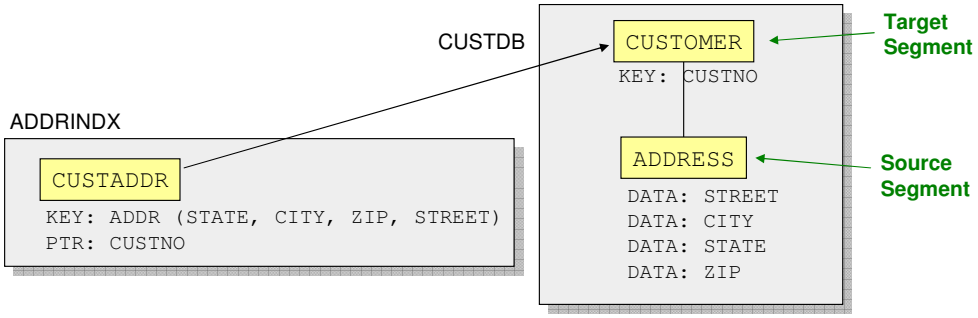
Fast Path Secondary Indexes

Fast Path Secondary Indexes

- IMS 12 adds support for secondary indexes with DEDBs
  - Secondary indexes are full function databases (HISAM or SHISAM)
  - Support for maintenance of secondary indexes
    - Index is updated when source segment is inserted, deleted, or replaced
  - No support for the creation of secondary indexes
    - A tool or program is required to add a secondary index to a database
- Benefits
  - New and simplified programming opportunities with DEDBs
    - Allows alternate sequencing of data
    - Allows alternate key access to data
  - Access to data in secondary index without accessing the DEDB

Fast Path Secondary Indexes

- Secondary indexes are either HISAM or SHISAM databases
  - Non-unique keys require HISAM
- Secondary index key is built from 1 to 5 fields in source segment
- Pointers are symbolic (concatenated key of target)
  - Reorganization of DEDB does not require changes to secondary indexes



691

Either HISAM or SHISAM may be used for Fast Path secondary indexes. If the secondary index may have non-unique keys, it must be HISAM.

The secondary index key is built from up to five fields in the source segment. These fields do not have to be adjacent in the source segment. They may be placed in any order in the secondary index key.

The target segment in the DEDB is the segment to which the secondary index points. The source segment in the DEDB is the segment which contains the fields used to build the secondary index entries. The source segment may be the target or it may be a dependent of the target.

The pointer from the secondary index to the DEDB is a symbolic pointer. That is, it is the concatenated key of the target segment. By using a symbolic key the secondary index is not affected by reorganizations of the DEDB.

In the example shown here, the DEDB is a customer database, CUSTDB. The root segment is the CUSTOMER segment. Its key is the customer number. The source segment for the secondary index is the ADDRESS segment. The key of the secondary index is built from four data fields. In the key they are ordered: state, city, zip and street. This is not the order of the fields in the address segment. The target segment is CUSTOMER. The secondary index, ADDRINDX, points to this segment using a symbolic pointer. This means that the pointer is the key of the CUSTOMER segment which is the parent of the source segment from which the key was built.

Fast Path Secondary Indexes

- Target can be any segment other than an SDEP
  - Target and its parents must have keys
- Source segment can be target or dependent under target
  - Exception: SDEPs cannot be source segments
- Up to 32 secondary indexes per target segment
- Up to 255 secondary indexes per database
- Sparse indexing is supported
  - Does not create index entries for some source segments
    - Determined by exit routine or NULLVAL value specified in DBD

692

Any segment other than an SDEP may be the target of a secondary index. The source segment may be the target segment or a dependent of the target with one exception. An SDEP cannot be a source segment. This restriction exists because of the way that SDEPs are deleted. They are deleted with the Delete utility which does not read the segments. This does not allow IMS to update the secondary index when SDEPs are deleted.

When a dependent segment is the target, it and its parents must have keys. For example, if A is the root, B is a dependent of A, C is a dependent of B and C is the target segment, A, B and C must have keys. Keys are required because the pointers from the secondary index are symbolic. That is, a pointer is the concatenated key of the target segment.

A segment may be the target of up to 32 secondary indexes. A database may have up to 255 secondary indexes pointing to it.

Sparse indexing is supported. When sparse indexing is used, index entries are not created for some source segments. This may be controlled in one of two ways. One may specify a NULLVAL value for which secondary index entries are not created. Typically, this is done when the field has a value of zero or blanks. Alternatively, one may use a Secondary Index Database Maintenance exit routine for sparse indexing. The exit routine is passed the source segment and determines if an index entry should be built for the segment. If you use both the NULLVAL and the exit routine, the entry is not built if either indicates it should not be built.

Indexing When Target Is Root Segment

- All segments under the root are accessible
  - Including SDEPs
- Key feedback area contents:
  - For target (root segment)
    - IMS returns the key of the pointer segment
      - That is, the secondary index key
  - For dependent segments
    - IMS returns key of the pointer segment plus the key of the dependent segment(s)

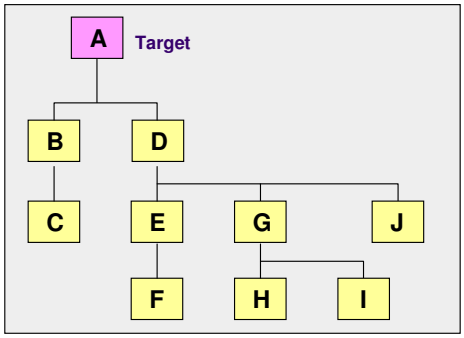
693

When the target segment is the root segment, all segments in the DEDB are accessible through the secondary index. This includes SDEP segments. The key feedback area uses the secondary index key for the key of the target segment (the root). The key feedback area uses the dependent segment keys when they are accessed. The concatenated key in the key feedback area is composed of the secondary index key and the keys of the dependent segments.

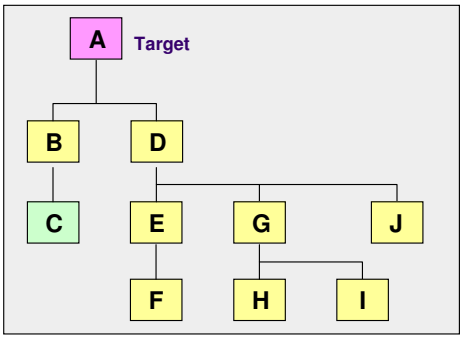
Indexing When Target Is Root Segment

- Indexing on a root segment

Physical data structure



Secondary data structure



Key Feedback for C:
Secondary index key + key of B + key of C

694

This page illustrates the physical structure of a database and the structure as viewed when accessing the database through the secondary index. In this case, they are the same since the root segment is also the target segment. As explained on the previous page, the key feedback area is composed of the secondary index key and the keys of the dependent segments. The key feedback area for segment C is composed of the secondary index key, the key of segment B and the key of segment C.

Indexing When Target Is Dependent Segment

- All direct parent segments of the target up to the root are accessible
- All children segments of the target are accessible
  - Full function also allows access to all children of the root
- SDEPs are not accessible through index
  - They are dependents of root and have no children
- Key feedback area contents:
  - For target
    - IMS returns the key of the pointer segment
      - That is, the secondary index key
  - For dependent segments
    - IMS returns key of the pointer segment plus the key of the dependent segment(s)

695

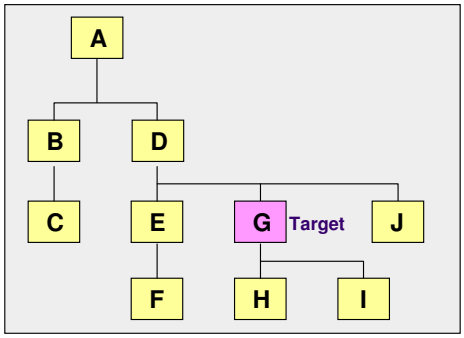
When the target segment is not the root, all parents (grandparents, etc.) are accessible through the secondary index. So are the dependents of the target segment. Fast Path is different from full function. Full function allow access to all children of the root segment, even those that are not direct parents or dependents of the target segment. When the target segment is not the root, SDEPs are not accessible through the secondary index. SDEPs are dependents of the root. They have no children and cannot be source segments. This means that they cannot be target segments.

When accessing a segment through the secondary index, the key feedback area uses the secondary index key for the key of the target segment. The key feedback area uses the keys of the other segments when they are accessed. The concatenated key in the key feedback area is composed of the secondary index key and the keys of the other segments in the path from the secondary index.

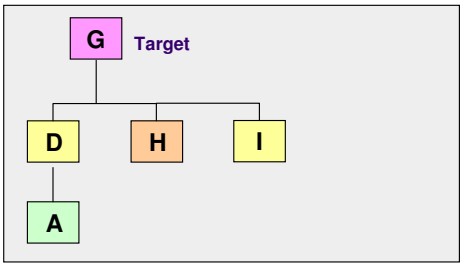
Indexing When Target Is Dependent Segment

- Indexing on a root segment

Physical data structure



Secondary data structure



Key Feedback for A:
Secondary index key + key of D + key of A

Key Feedback for H:
Secondary index key + key of H

696

This page illustrates the physical structure of a database and the structure as viewed when accessing the database through the secondary index. In this case, the target is not the root segment. The target is segment G.

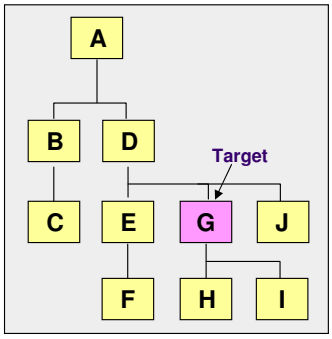
The structure as seen from the secondary index path has the target, segment G as the root. Segment D is the parent of G in the physical structure, so it is the first dependent in the secondary structure. Its parent is A so it is a dependent of D. Segments H and I are the only children of segment G in the physical structure. They are also children in the secondary structure.

The illustration shows the key feedback areas for segments A and H. The key feedback area for A includes the secondary index key, the key of segment D and the key of segment A. The key feedback area for H includes the secondary index key and the key of segment H.

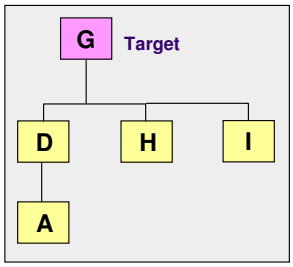
SENSEG Statements in PSB

- SENSEG statements are defined in the physical structure order
  - This is different from full function
    - Full function SENSEG statements are defined in the access order

Physical data structure



Secondary data structure



PSB

```
PCB  TYPE=DB,DBNAME=,..,
  PROCSEQD=…
SENSEG NAME=A,PARENT=0
SENSEG NAME=D,PARENT=A
SENSEG NAME=G,PARENT=D
SENSEG NAME=H,PARENT=G
SENSEG NAME=I,PARENT=G
PSBGEN   PSBNAME=…,
END
```

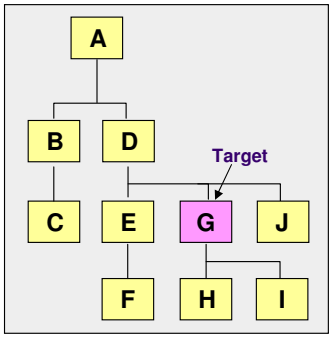
697

When accessing a database through the secondary index the SENSEG statements in PSB must be in the physical structure order. This is different from the requirements for full function databases. With full function databases the SENSEG statements must be in the secondary structure order. In this illustration the secondary structure order is G, D, A, H and I. Nevertheless, the SENSEG statement order is A, D, G, H and I. This matches the physical structure order for these segments.

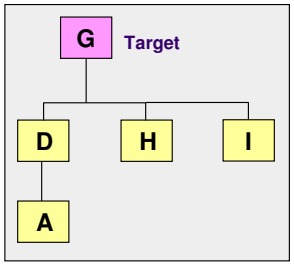
SENSEG Statements and Access Order

- Even though SENSEG statements are defined in the physical structure order access is in secondary data structure order
  - Unqualified GN calls return:
    - G, D, A, H and I segments in that order
  - PCB does not provide the programmer with segment access order

Physical data structure



Secondary data structure



PSB

```
PCB TYPE=DB,DBDNAME=,..,
  PROCSEQD=…
SENSEG NAME=A,PARENT=0
SENSEG NAME=D,PARENT=A
SENSEG NAME=G,PARENT=D
SENSEG NAME=H,PARENT=G
SENSEG NAME=I,PARENT=G
PSBGEN  PSBNAME=…,
END
```

698

Even though the SENSEG statements in the PSB are in physical structure order, the segments are accessed in the secondary structure order when accessing the database through the secondary index. For example, a program using unqualified GN calls would access segments in the order G, D, A, H and I. This does not match the order of the SENSEG statements which is A, D, G, H and I.

This difference between the order of the segments in the PCB and the order in which the programmer accesses them is unique with DEDB secondary indexes.

DBDGEN for DEDB with Secondary Index

- DBDGEN for DEDB includes LCHILD and XDFLD statements
- LCHILD statement follows SEGM statement for the target
  - It names the secondary index database and segment
  - It must specify PTR=SYMB

```
LCHILD NAME=(sisegname,sidbname),PTR=SYMB
```

699

To include secondary indexes for a DEDB you must specify an LCHILD and XDFLD statement for each secondary index.

The LCHILD statement follows the SEGM statement for the segment which is the target of the secondary index. The NAME= parameter specifies the segment in the secondary index and the secondary index database name. We will see later that multiple secondary index databases are specified when using User Data Partitioning. The PTR= parameter must specify SYMB. This indicates that the pointer is symbolic. That is, the pointer is the concatenated key of the target segment.

DBDGEN for DEDB with Secondary Index

- XDFLD statement follows the LCHILD statement
  - It defines the field(s) on which index is built (SRCH=)
    - The fields may be in the target or a segment which follows in the hierarchy
    - All fields must be in the same segment (SEGMENT=sourcesegment)
  - It may include duplicate data fields (DDATA=)
  - It may specify subsequence fields (SUBSEQ=)
  - It may specify index suppression (NULLVAL= and/or EXTRTN=)
  - The XDFLD statement includes a name which may be used in SSAs to search on the secondary index (NAME=)

```
XDFLD  NAME=searchname,SEGMENT=segmname,
       SRCH=(fldname1,fldname2,…),
       DDATA=(fldnamea,fldnameb,…),
       SUBSEQ=(fldnamex,fldnamey,…),
       NULLVAL=value,EXTRTN=rtnname
```

700

The XDFLD statement follows the LCHILD statement. The XDFLD statement defines the field or fields on which the secondary index is built. If the source segment is not the target segment, it is specified in the SEGMENT= parameter. The source segment may be either the target segment or one of its dependents. If SEGMENT= is not included, the source segment is the target segment. The field or fields on which the secondary index is built are defined in the SRCH= parameter. Up to five fields may be specified. All fields must be in the same segment. The DDATA=, SUBSEQ=, and NULVAL= parameters are optional.

DDATA= defines duplicate data fields. These are fields in the source segment that are also included in the secondary index segment. They are available to programs which process the secondary index as a database.

SUBSEQ= defines subsequence fields. These are fields which are used to sequence secondary index segments which would have the same secondary index key. Concatenated key fields (/CKxxxxx) may be specified as SUBSEQ= fields. This is explained later.

Index suppression is optional. It may be specified in two ways, with the NULLVAL= parameter and with the EXTRTN= parameter. Either, neither or both may be specified. NULLVAL= is a one byte field. If it is specified, a SRCH= value which has this value is all of its bytes will cause index suppression. Index suppression causes no index entry to be created for the source segment. If there are multiple fields specified in SRCH=, all must have this value for index suppression to be invoked. EXTRTN= specifies a Secondary Index Database Maintenance exit routine. The exit routine is passed the source segment and determines if an index entry should be built for the segment.

The NAME= parameter specifies the name which may be used in SSAs to qualify calls on the value of the secondary index.

DBDGEN for Secondary Index

- DBDGEN for secondary index
  - Requires INDEX VSAM or SHISAM ACCESS= parameter on DBD statement
    - ACCESS=(INDEX,VSAM) is used when HISAM database is index

```
DBD NAME=indexname,ACCESS=(INDEX,VSAM),FPINDEX=YES
```

    - ACCESS=(INDEX,SHISAM) is used when SHISAM database is index

```
DBD NAME=indexname,ACCESS=(INDEX,SHISAM),FPINDEX=YES
```

- HISAM supports non-unique index keys
  - Non-unique keys require overflow data set specification on DATASET statement

```
DATASET  DD1=ddname1,OVFLW=ddname2
```

- SHISAM supports only unique index keys
  - Overflow data set is never specified on DATASET statement

```
DATASET  DD1=ddname1
```


The DBD for the secondary index requires ACCESS=(INDEX,VSAM) for a HISAM secondary index or ACCESS=(INDEX,SHISAM) for a SHISAM secondary index. Note that you do not specify ACCESS=(HISAM,VSAM) for HISAM or ACCESS=(SHISAM,VSAM) for SHISAM secondary indexes.

The FPINDEX=YES parameter is optional. It sets a flag in the DMB. This flag may be used by Fast Path tools. Some tools may require that it be specified.

HISAM secondary indexes support non-unique keys. They require an overflow data set. When using non-unique keys the DATASET statement includes both the DD1= paremeter and the OVFLW= parameter. They specify the DD names for these data sets. In unique keys are used the DATASET statement specifies only the DD1= parameter. Unique keys are supported with both HISAM and SHISAM.

DBDGEN for Secondary Index

- SEGM statement

```
SEGM NAME=segmname,PARENT=0,BYTES=segsize
```

- FIELD statement

```
FIELD NAME=(fldname,SEQ,U|M),BYTES=fldsize,START=1
```

- LCHILD statement
  - Pointer must be symbolic (PTR=SYMB)

```
LCHILD NAME=(targetsegm,targetdb),INDEX=xdfldname,PTR=SYMB
```


There is only one SEGM statement for the secondary index. It specifies the segment name, PARENT=0, and the size (BYTES=) of the segment. The segment size includes the size of the secondary index key plus the size of the concatenated key of the target segment. It also includes the size of any subsequence and duplicate data fields. It may be larger than the sum of these fields. If it is larger, the remaining space may be used for user data.

One or more FIELD statements must be included. The secondary index key is defined with SEQ included in the NAME= parameter. If the key including the subsequence field is unique, specify U. If it is not unique, specify M. The BYTES= parameter specifies the size of the sequence field. This includes the search field(s) and subsequence field(s). START=1 should always be specified.

The LCHILD statement NAME= parameter specifies the target segment name and the target database. The INDEX= parameter value must match what is specified in the NAME= parameter on the XDFLD statement in the target database. PTR=SYMB must be specified.

DBDGEN for Secondary Index

- Unique vs. non-unique keys for secondary index
  - Unique and non-unique keys are supported with HISAM secondary indexes
  - Only unique keys are supported with SHISAM secondary indexes
  - Unique keys may be created with SUBSEQ= on XDFLD statement
    - SUBSEQ= may specify up to five fields
  - Unique keys may be created by specifying a FIELD statement with name of /CKxxxxx and including it on SUBSEQ= on XDFLD statement
    - Used to indicate that key includes concatenated key of source segment
  - /SX fields may not be used with Fast Path secondary indexes
- Duplicate data in secondary index is supported
  - DDATA= specified on XDFLD statement
  - DDATA= may specify /CK field (concatenated key)

Secondary indexes may have either unique or non-unique keys. HISAM supports both unique and non-unique keys. SHISAM requires unique keys. Subsequence fields (SUBSEQ= on the XDFLD statement) may be used to create unique keys. One way to create unique keys is using the concatenated key of the source segment as a subsequence field. This is done by including a FIELD statement in the DEDB segment whose name begins with /CK and including this field in the SUBSEQ= parameter of the XDFLD statement. The /CK field does not guarantee uniqueness if the keys of the source segment and its parents are not unique.

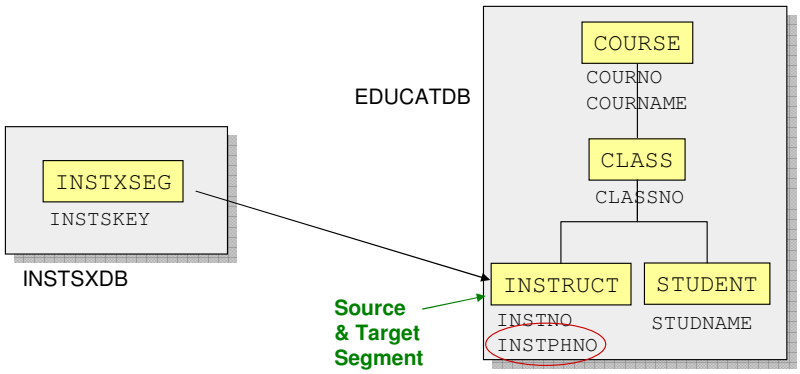
Secondary indexes for full function databases may use a /SX field to create unique keys. The /SX field is either the four-byte relative byte address (RBA) of the source segment or the 8-byte indirect list key (ILK) of the segment. The ILK is used by HALDB. The /SX field cannot be used with DEDBs. DEDB segments do not have an ILK and their RBA may be changed by a replace (REPL) call.

Duplicate data may be included in a secondary index by including DDATA= on the XDFLD statement. Duplicate data is only available to application programs when they process the secondary index as a database. It is not available to them when they use the secondary index to process the DEDB.

SHISAM does not support CI reclaim with data sharing; therefore, when all index entries from a large range of keys may be deleted, data sharing users may prefer to choose HISAM even when the secondary index has unique keys.

DBDGEN Example 1 – Target Segment = Source Segment

- Example of secondary index
  - Secondary index is HISAM
  - Target segment is the source segment
  - Secondary index keys are non-unique



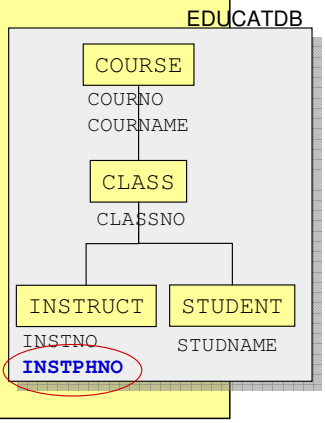
704

This is an example of a secondary index where the target segment is also the source segment. Segment INSTRUCT in the EDUCATDB database is the source and target segment. The index is built on the INSTPHON field. The next pages show the DBDs for the DEDB and the secondary index.

DBDGEN Example 1 – Target Segment = Source Segment

- DBD for indexed database
  - Target segment INSTRUCT is also the source segment

```
DBD1    DBD  NAME=EDUCATDB,ACCESS=DEDB,RMNAME=RMOD3
AREA    DD1=EDAREA1,SIZE=1024,UOW=(100,10),ROOT=(236,36)
SEGM    NAME=COURSE,PARENT=0,BYTES=100
FIELD   NAME=(COURNO,SEQ,U),BYTES=5,START=1
FIELD   NAME=COURNAME,BYTES=10,START=15
SEGM    NAME=CLASS,BYTES=50,PARENT=COURSE
FIELD   NAME=(CLASSNO,SEQ,U),BYTES=4,START=7
SEGM    NAME=INSTRUCT,BYTES=25,PARENT=CLASS
FIELD   NAME=(INSTNO,SEQ,U),BYTES=6,START=1
FIELD   NAME=INSTPHNO,BYTES=10,START=11
LCHILD  NAME=(INSTXSEG,INSTSXDB),PTR=SYMB
XDFLD   NAME=XINST,SRCH=INSTPHNO
SEGM    NAME=STUDENT,BYTES=40,PARENT=CLASS
FIELD   NAME=STUDNAME,BYTES=20,START=1
DBDGEN
```



705

This is the DBD for the DEDB. Segment INSTRUCT is the source and the target segment. The LCHILD and XDFLD statements follow the SEGM statement for INSTRUCT.

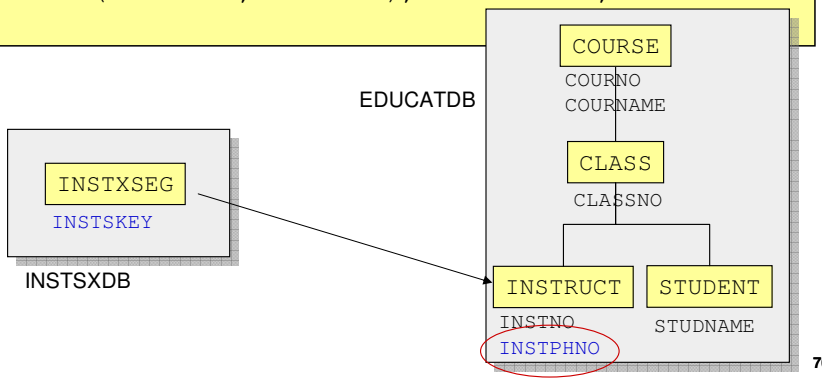
The LCHILD segment specifies the secondary index segment, INSTXSEG, and database, INSTSXDB. PTR=SYMB is specified, as required.

The XDFLD statement specifies the name of the search field, XINST, for use with the secondary index. It also specifies that field INSTPHON is used to build the search field.

DBDGEN Example 1 – Target Segment = Source Segment

- DBD for secondary index database
  - HISAM database

```
DBDSX    NAME=INSTSXDB,ACCESS=(INDEX,VSAM)
DATASET  DD1=INSTKSDS,OVFLW=INSTOVFL
SEGM     NAME=INSTXSEG,PARENT=0,BYTES=24
FIELD    NAME=(INSTSKEY,SEQ,M),BYTES=10,START=1
LCHILD   NAME=(INSTRUCT,EDUCATDB),INDEX=XINST,PTR=SYMB
DBDGEN
```



This is the DBD for the secondary index database. It is HISAM, therefore, ACCESS=(INDEX,VSAM) is specified.

DD1= on the DATASET statement specifies the DD name of the primary secondary index data set. Since the key is not unique, the OVFLW= parameter is specified. It indicates the DD name of the overflow data set.

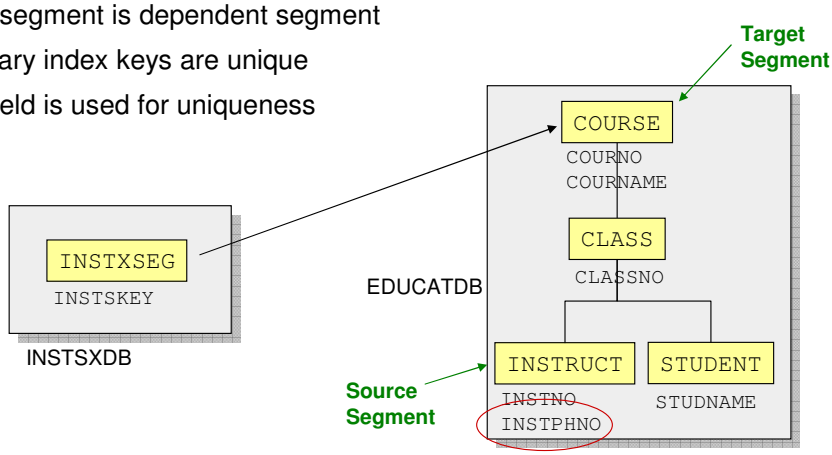
The SEGM statement defines the segment in the secondary index. Its size is 24 bytes. This is the size of the INSTPHNO field which is 10 bytes plus the size of the concatenated key of the INSTRUCT segment. The concatenated key is 15 bytes. It is composed of 5 bytes for the COURSE segment, 4 bytes for the CLASS segment and 6 bytes for the INSTRUCT segment.

The FIELD statement defines the sequence field in the secondary index. Since the keys are not unique, M is specified in the NAME= parameter. The size of the sequence field is 10 bytes. This is the size of the INSTPHNO field in the INSTRUCT segment which is the source segment.

The LCHILD statement specifies the target segment, INSTRUCT, and database, EDUCATDB, in the NAME= parameter. The INDEX= parameter specifies the NAME= value on the XDFLD statement of the target database. This is XINST, the search field for use with the secondary index. PTR=SYMB is required for Fast Path secondary index databases.

DBDGEN Example 2 – Target Segment ≠ Source Segment

- Example of secondary index
  - Secondary index is HISAM
  - Target segment is the root segment
  - Source segment is dependent segment
  - Secondary index keys are unique
    - /CK field is used for uniqueness



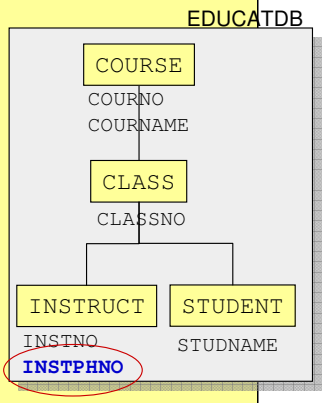
707

This is an example of a secondary index where the source segment is not the target segment. The COURSE segment is the target. The INSTRUCT segment is the source. The index is built on the INSTPHON field. In this example, a /CK field is used to create unique keys. The next pages show the DBDs for the DEDB and the secondary index.

DBDGEN Example 2 - Target Segment ≠ Source Segment

- DBD for indexed database
  - Target segment is COURSE; Source segment is INSTRUCT

```
DBD1    DBD  NAME=EDUCATDB,ACCESS=DEDB,RMNAME=RMOD3
AREA    DD1=EDAREA1,SIZE=1024,UOW=(100,10),ROOT=(236,36)
SEGM    NAME=COURSE,PARENT=0,BYTES=100
FIELD   NAME=(COURNO,SEQ,U),BYTES=5,START=1
FIELD   NAME=COURNAME,BYTES=10,START=15
LCHILD  NAME=(INSTXSEG,INSTSXDB),PTR=SYMB
XDFLD   NAME=XINST,SEGMENT=INSTRUCT,
        SRCH=INSTPHNO,SUBSEQ=/CKAAAAA
SEGM    NAME=CLASS,BYTES=50,PARENT=COURSE
FIELD   NAME=(CLASSNO,SEQ,U),BYTES=4,START=7
SEGM    NAME=INSTRUCT,BYTES=25,PARENT=CLASS
FIELD   NAME=(INSTNO,SEQ,U),BYTES=6,START=1
FIELD   NAME=INSTPHNO,BYTES=10,START=11
FIELD   NAME=/CKAAAAA,BYTES=15,START=1
SEGM    NAME=STUDENT,BYTES=40,PARENT=CLASS
FIELD   NAME=STUDNAME,BYTES=20,START=1
DBDGEN
```



708

This is the DBD for the DEDB. Segment COURSE is the target segment. INSTRUCT is the source segment. The LCHILD and XDFLD statements follow the SEGM statement for COURSE.

The LCHILD segment specifies the secondary index segment, INSTXSEG, and database, INSTSXDB. PTR=SYMB is specified, as required.

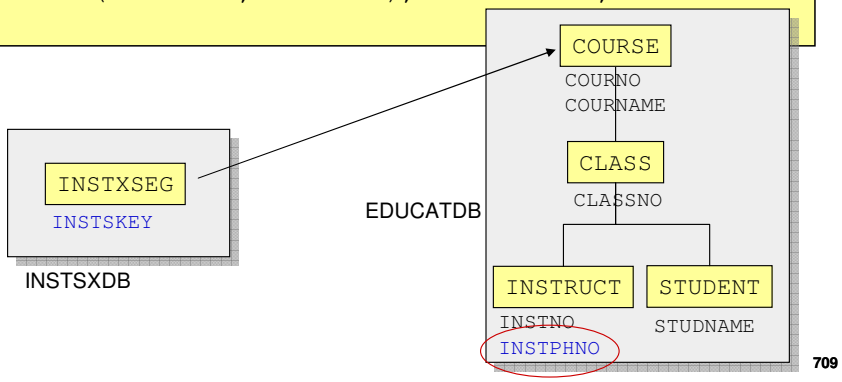
The XDFLD statement specifies the name of the search field, XINST, for use with the secondary index. Since the source segment is not the target segment, SEGMENT=INSTRUCT is specified to indicate that INSTRUCT is the source segment. SRCH=INSTPHON specifies that field INSTPHON is used to build the search field. SUBSEQ=/CKAAAAA is used to create a unique key.

A FIELD statement with NAME=/CKAAAAA is included so that the XDFLD SUBSEQ= parameter may reference it.

DBDGEN Example 2 - Target Segment ≠ Source Segment

- DBD for secondary index database
  - HISAM database

```
DBDSX    DBD   NAME=INSTSXDB,ACCESS=(INDEX,VSAM)
DATASET  DD1=INSTKSDS
SEGM     NAME=INSTXSEG,PARENT=0,BYTES=30
FIELD    NAME=(INSTSKEY,SEQ,U),BYTES=25,START=1
LCHILD   NAME=(INSTRUCT,EDUCATDB),INDEX=XINST,PTR=SYMB
DBDGEN
```



This is the DBD for the secondary index database. It is HISAM, therefore, ACCESS=(INDEX,VSAM) is specified.

DD1= on the DATASET statement specifies the DD name of the secondary index data set. Since the key is unique, the OVFLW= parameter is not specified on the DATASET statement.

The SEGM statement defines the segment in the secondary index. Its size is 30 bytes. It is composed of the secondary index key (10 bytes), the concatenated key of the INSTRUCT segment (/CKAAAAA field which is 15 bytes) and the symbolic pointer to the target (5 bytes).

The FIELD statement defines the sequence field in the secondary index. Since the keys are unique, U is specified in the NAME= parameter. The size of the sequence field is 25 bytes. This is the size of the INSTPHNO field (10 bytes) in the INSTRUCT segment and the /CKAAAAA field (15 bytes).

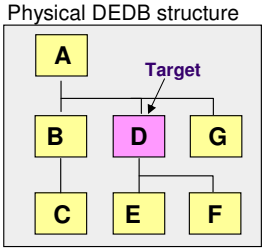
The LCHILD statement specifies the target segment, COURSE, and database, EDUCATDB, in the NAME= parameter. The INDEX= parameter specifies the NAME= value on the XDFLD statement of the target database. This is XINST. PTR=SYMB is required for Fast Path secondary index databases

PSBGEN with Fast Path Secondary Index

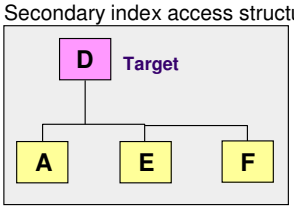
- Use of secondary index requires PROCSEQD= on PCB
  - Indicates that the specified secondary index is used to access the database
- SENSEGs are in physical database order

```
PCB TYPE=DB,DBDNAME=DEDB12,PROCOPT=G,KEYLEN=30,PROCSEQD=SINDX12
SENSEG NAME=A,PARENT=0
SENSEG NAME=D,PARENT=A
SENSEG NAME=E,PARENT=D
SENSEG NAME=F,PARENT=G
PSBGEN LANG=COBOL,PSBNAME=JWS123
END
```

Physical DEDB structure



Secondary index access structure



KEYLEN must be larger of:

- Largest concatenated key in physical structure
- Largest concatenated key in secondary structure

710

When you use a Fast Path secondary index you must specify PROCSEQD= on the PCB statement in the PSB. Note that this is “PROCSEQD”, not PROCSEQ”. The “D” is added to indicate that the index is for a DEDB. As was mentioned earlier, the order of the SENSEG statements is the physical order in the indexed database, not the secondary index processing order. In this example, the SENSEG statements appear in the order A, D, E and F. The secondary index processing order is D, A, E and F.

The KEYLEN value on the PCB statement must be the larger of:

1. The largest concatenated key in the physical structure for any segment which might be accessed by the secondary index. In this case, it is the larger of the concatenated keys for segments E and F.
2. The largest concatenated key in the secondary structure. The key of the target segment in the secondary structure is the secondary index key. In this case, the target is segment D. The largest concatenated key is the secondary index key plus the largest of the keys for segments A, E and F.

The INDICES= parameter is not supported for Fast Path secondary indexes. INDICES= allows the use of the secondary index in SSAs without using the secondary index for alternate access to the database.

Secondary Index Capabilities Unique to Fast Path

- User data partitioning
  - Secondary index may be spread across multiple index databases
  - Supports very large indexes
- Multiple secondary index segments
  - One index may be based on different search fields
    - Search fields must be in the same source segment
    - Search fields must be the same size
  - Allows the building of one index for different fields with similar data
    - e.g. phone numbers (home phone, work phone, cell phone,…)

Secondary indexes for Fast Path databases have some capabilities that are not available with secondary indexes for full function databases.

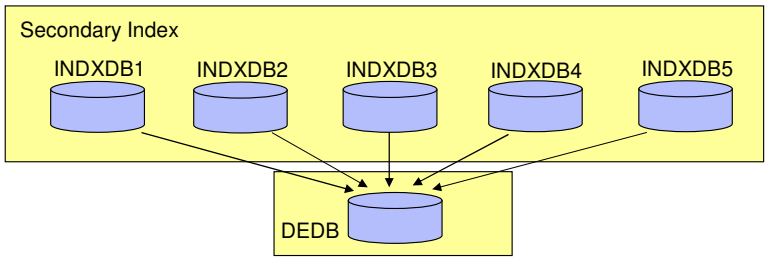
The first of these capabilities is user data partitioning. This allows a secondary index to be spread across multiple physical databases. Obviously, this supports very large indexes.

The second of these capabilities in the support for “multiple secondary index segments.” This is one index created from different fields in the same source segments. Since the index has one key size, the search fields must be the same size. This capability allows you to build one index from similar data which is stored in different fields. An example is an index based on telephone numbers. Multiple phone numbers, such as home phone, work phone, and cell phone could be stored in different fields, but only one index is used. The index would have an entry for each phone number.

These capabilities are explained in more detail on the following pages.

User Data Partitioning

- User data partitioning
  - Secondary index may be spread across multiple index databases
    - Each index database contains a range of keys
    - HISAM or SHISAM may be used
  - Requires user partition selection exit routine
    - Routine assigns index entry to an index database
      - Based on secondary index key
  - Multiple index databases (partitions) must have same structure and attributes



712

User data partitioning allows a secondary index to be spread across multiple physical databases. Each index database contains a range of keys. Either HISAM or SHISAM may be used, but all “partitions” in an index must be the same type. Index keys are assigned to an index database by a user partition selection exit routine. The index is passed the secondary index key. Each of the databases in an index must have the same structure and attributes. They may have different sizes to accommodate the number of entries that could exist in the different key ranges.

User Data Partitioning

- DBD for User Data Partitioning
  - Specify multiple index databases on DEDB LCHILD statement

    ```
    LCHILD NAME=(segname,(db1name,db2name,…)),PTR=SYMB
    ```

    - Order of databases determines order of partitions for get next processing
  - Specify PSELRTN= on DEDB XDFLD statement
  - Optionally specify PSELOPT= on DEDB XDFLD statement

    ```
    XDFLD  NAME=searchname,SRCH=fldname,PSELRTN=rtnname,
           PSELOPT=MULT|SNGL,…
    ```

- Partition Selection Routine
  - Routine name specified on PSELRTN= parameter of DEDB XDFLD statement
  - Called when an insert or qualified get call is issued
    - Used to determine which secondary index database is used for call
  - Routine may be shared by multiple secondary indexes on multiple DEDBs

713

User data partitioning is defined by specifying multiple database names on the LCHILD statement in the indexed database. The order of the databases in the LCHILD statement determines the order in which the index partitions are processed by get next calls. This means that a get next call which reaches the end of an index database will continue to the next database defined in the LCHILD statement.

User data partitioning also requires the specification of PSELRTN= on the XDFLD statement in the indexed database. This parameter specifies the user partition selection exit routine.

You may specify the PSELOPT= parameter on the XDFLD statement. This partition selection option is explained on the next page.

The partition selection routine is called when an insert or qualified get call is issued. The routine determines which secondary index database is used for the call. That is, it is used to specify in which database a secondary index key is stored. The routine may be shared by multiple secondary indexes. These indexes may be on different databases.

A sample partition selection routine, DBFPSE00, is supplied by IMS.

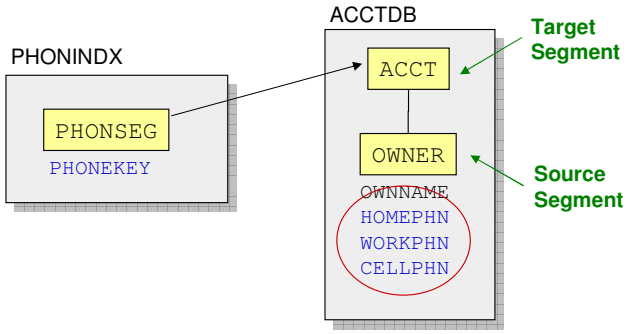
User Data Partitioning

- Partition selection option (PSELOPT=) parameter
  - Determines when 'GB' status code is returned for get next call
    - PSELOPT=MULT returns 'GB' at the end of the secondary index
      - At the end of the last user partition in the secondary index
    - PSELOPT=SNGL returns 'GB' at the end of the secondary index partition
      - At the end of the current user partition in the secondary index
  - Only used with secondary index
    - PROCSEQD= specified on the PCB statement in the PSB
      - First partition is specified
  - Specified on DBD XDFLD statement in DEDB and/or on the PCB
    - If specified on both, the PCB specification is used
    - If not specified on the PCB, the DBD specification is used
    - Default for DBD is PSELOPT=MULT

The partition selection option controls whether only one secondary index user partition is used for a call. It is only used with secondary index processing. This means that PROCSEQD= must be specified in the PCB used for the call. The value specified for PROCSEQD= must be the first partition. PSELOPT=MULT indicates that all partitions are used. A 'GB' (end of database) status code is returned when the end of the last secondary index partition is reached. PSELOPT=SNGL indicates that only one partition is used. A 'GB" status code is returned if the call reaches the end of the current partition. PSELOPT= may be specified in two places, the DBD and the PCB. The specification in the PCB overrides the specification in the DBD. If neither is specified, PSELOPT=MULT is used.

Multiple Secondary Index Segments

- Multiple secondary index segments
  - One index may be based on different search fields
    - Each search field creates index segment occurrence
    - Search fields must be in the same source segment
    - Search fields must be the same size
    - Example
      - Secondary index with entries for home, work and cell phones



715

Multiple secondary index segments is the support for one index created from different fields in the same source segments. Each search field (or set of search fields) is used to create an entry in the secondary index. These fields or sets of fields must be the same size. In this example, there are three fields for telephone numbers in the OWNER segment. The index would contain an entry for each of the phone numbers.

Multiple Secondary Index Segments

- Definition of multiple secondary index source segments
  - Multiple LCHILD/XDFLD-statement pairs in the DBD
    - Under SEGM statement of a target segment from a single source segment
  - LCHILD statement must include MULTISEG=YES
    - MULTISEG=NO is the default

```
LCHILD NAME=(sisegname,sidbname),PTR=SYMB,MULTISEG=YES
```

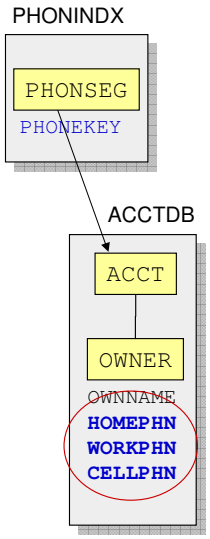
716

Multiple secondary index segments are defined by including multiple LCHILD and XDFLD statement pairs in the indexed database. The LCHILD segment must include the MULTISEG=YES parameter. MULTISEG=NO is the default for this parameter.

Multiple Secondary Index Segments

- Definition of multiple secondary index source segments

```
DBD     NAME=ACCTDB,ACCESS=DEDB,RMNAME=RMD4
AREA    DD1=ACCT1,SIZE=1024,UOW=(100,10),
        ROOT=(236,36)
SEGM    NAME=ACCT,PARENT=0,BYTES=100
FIELD   NAME=(ACCTNO,SEQ,U),BYTES=12,START=1
LCHILD  NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
XDFLD   NAME=XPHON,SEGMENT=OWNER,SRCH=HOMEPHN
LCHILD  NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
XDFLD   NAME=XPHON,SEGMENT=OWNER,SRCH=WORKPHN
LCHILD  NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
XDFLD   NAME=XPHON,SEGMENT=OWNER,SRCH=CELLPHN
SEGM    NAME=OWNER,BYTES=300,PARENT=ACCT
FIELD   NAME=OWNNAME,BYTES=40,START=1
FIELD   NAME=HOMEPHN,BYTES=10,START=41
FIELD   NAME=WORKPHN,BYTES=10,START=51
FIELD   NAME=CELLPHN,BYTES=10,START=61
DBDGEN
```



717

This is an illustration of coding a DBD for a DEDB with a secondary index using the multiple secondary index segment capability. There are multiple LCHILD and XDFLD statements which refer to the same secondary index. The LCHILD statements are the same. The XDFLD statements have the same NAME= and SEGMENT= parameters. The SRCH= parameters point to the different fields which are used for indexing.

The secondary index DBD for this example is:

```
DBDSX    DBD   NAME=PHONINDX,ACCESS=(INDEX,VSAM)
               DATASET DD1=PHONKSDS,OVFLW=PHONOVFL
               SEGM    NAME=PHONSEG,PARENT=0,BYTES=22
               FIELD   NAME=(PHONEKEY,SEQ,U),BYTES=10,START=1
               LCHILD  NAME=(OWNER,ACCTDB),INDEX=XPHON,PTR=SYMB
               DBDGEN
```

In this example the XDFLD statements in the DEDB DBD would probably include NULLVAL=' ' parameters to specify that a secondary index entry is not built when the phone field contains blanks. These NULLVAL parameters are not included on this visual due to the space limitations for the example.

Secondary Index Segment Fields

- HISAM Secondary Index with Unique Keys

Delete Byte	Search (Key)	Subsequence (Optional)	Duplicate Data (Optional)	Symbolic Pointer (Concatenated Key)	User Data (Optional)

- HISAM Secondary Index with Non-unique Keys

Duplicate Key Pointer	Delete Byte	Search (Key)	Subsequence (Optional)	Duplicate Data (Optional)	Symbolic Pointer (Concatenated Key)	User Data (Optional)

- SHISAM Secondary Index

Search (Key)	Subsequence (Optional)	Duplicate Data (Optional)	Symbolic Pointer (Concatenated Key)	User Data (Optional)

This page documents the layout of Fast Path secondary index segments and their fields.

There is not a segment code field in the HISAM segments. This is unlike other HISAM databases. They always have a one-byte segment code field preceding the delete byte.

The Search (Key) field is the key of the secondary index. If the subsequence field is present it is included with the search field in the VSAM key length.

The duplicate key pointer field in the HISAM secondary index with non-unique keys is a four byte direct pointer to the overflow (ESDS) data set. The overflow data set contains more pointers with the same format. They are chained in last-in-first-out (LIFO) sequence.

The pointer to entries in the DEDB is always a symbolic pointer. That is, it is the concatenated key of the target segment.

User data fields are not defined directly in the DBD. They are defined by making the segment larger than the combined size of the search, subsequence, duplicate data and symbolic pointer fields.

Adding a Secondary Index to a DEDB

- Steps to add a secondary index to a DEDB
  - Modify DEDB DBD
    - Add LCHILD and XDFLD statements
  - Create secondary index DBD
  - Allocate secondary index data set(s)
  - Create PSB(s) with PROCSEQD= in PCB
  - DBDGEN, PSBGEN and ACBGEN
  - Take DEDB offline
  - Run tool or user program to create the secondary index
  - Add secondary index to system definition
    - CREATE DB command or system definition and online change
  - Add any new programs and transactions which use the secondary index
    - CREATE commands or system definition and online change
  - Online change for ACBLIB
  - Start DEDB

719

These are the steps to add a secondary index to an existing DEDB. Note that there are no utilities provided by IMS to add the secondary index. It must be done either with a tool or with a user written program to create the entries in a KSDS.

An alternative for adding a secondary index to a DEDB is shown later in this section.

Creating a New DEDB with a Secondary Index

- Steps to create a new DEDB with a secondary index
  - Create DEDB DBD with LCHILD and XDFLD statements
  - Create secondary index DBD
  - Allocate DEDB and secondary index data sets
  - Create PSB(s) with PROCSEQD= in PCB
  - DBDGEN, PSBGEN and ACBGEN
  - Add DEDB and secondary index to system definition
    - CREATE DB commands or system definition and online change
  - Add any new programs and transactions to system definition
    - CREATE commands or system definition and online change
  - Online change for ACBLIB
  - Run DEDB Initialization utility
  - Load the DEDB
    - This will create entries in the secondary index

These are the steps to create a new DEDB with a secondary index. In this case, a tool or user written program is not required to build the secondary index. Instead, the insertion of source segments in the DEDB causes the secondary index entries to be created.

Adding a Secondary Index to a DEDB without a Tool

- Create secondary index by loading the DEDB
  - Write user program to read all segments in the DEDB and write them to a file
  - Write user program to read the file and insert the segments into the DEDB
  - Run user read program
  - Take DEDB offline
  - Redefine DEDB with secondary index
  - Define Secondary Index
  - Delete and allocate the DEDB area data sets
  - Add program, transaction and database definitions to the online system
  - Start DEDB online
  - Run user insert program
    - Index maintenance will create secondary index entries

This is an alternative method for adding a secondary index to a DEDB. It does not require a tool. One can simply reload the database with a user program. First, you can read all the segments in the DEDB and write them to a user file. Then, read the file and insert the segments into the database. This is like the technique shown on the previous page for creating a new DEDB with a secondary index.

Accessing DEDB via Secondary Index

- Application uses PCB with DBDNAME=dedb,PROCSEQD=secindex

Example 1:

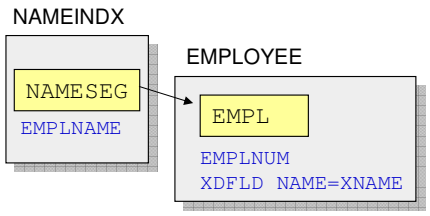
PCB with PROCSEQD=NAMEINDX

```
GU EMPL(XNAME=SMITH        )
```

Returns EMPL segment with name SMITH

```
GN EMPL
```

Returns EMPL segment with name SMITHERS which is the next name in alphabetical order



EMPLOYEE database with key of employee number (EMPLNUM).

Secondary index with key of employee name (EMPLNAME).

XDFLD statement has NAME=XNAME specified.

722

This is an illustration of an application's access via a secondary index. The key of the root segment is based on employee number. The secondary index is based on employee name. The program uses a PCB with PROCSEQD=NAMEINDX specified. This causes access to be via the secondary index. The first call is a GU qualified on the name field with a value of SMITH. A root with this name exists, so the segment with name value of SMITH is returned. The next call is a GN. It returns the next employee in the secondary index order. This is the employee with name SMITHERS.

Accessing Secondary Index as Database

- Secondary index may be accessed as a database
  - Does not require access to the DEDB
    - May be more efficient than accessing the DEDB
  - Either
    - Specify secondary index database name in DBDNAME= parameter and ACCESS=DB (default) of PCB statement

```
PCB TYPE=DB,DBDNAME=siname,…
```

    Or
    - On PCB statement specify DEDB name in DBDNAME= parameter, secondary index in PROCSEQD= parameter, and ACCESS=INDEX

```
PCB TYPE=DB,DBDNAME=dedbname,PROCSEQD=siname,ACCESS=INDEX,…
```

      - This technique is useful with user partitioning
        - All secondary index partitions are available to the program

723

The secondary index may be accessed as a database, not as a secondary index to the DEDB. This is typically done when there is duplicate data in the secondary index. If all the data that the application requires is in the secondary index, it is more efficient to access the secondary index as a database. This avoids accessing the DEDB.

Accessing the secondary index as a database may be done in either of two ways. First, one could specify the secondary index database name in the DBDNAME parameter on the PCB statement. Second, one may specify ACCESS=INDEX on the PCB statement with the DEDB name specified on the DBDNAME= parameter and the secondary index specified on the PROCSEQD= parameter. The ACCESS= parameter is new. It is used only when PROCSEQD= is specified. That is, it applies only to Fast Path secondary indexing. It is either ACCESS=DB or ACCESS=INDEX. ACCESS=DB is the default.

The second technique is especially useful with user partitioning. It gives the program access to all of the partitions in the secondary index. The PROCSEQD= parameter specifies the first partition, but the program has access to all partitions of the secondary index.

Accessing Secondary Index as Database

- When using

```
PCB TYPE=DB,DBDNAME=dedbname,PROCSEQD=siname,ACCESS=INDEX,…
```

  - GU, GHU, GN and GHN calls are allowed
  - ISRT calls are not allowed
  - DLET calls are not allowed
  - REPL calls are allowed
    - Search and subsequence fields may not be changed
    - All other fields may be changed
      - Changing user data fields is OK
      - **CAUTION:**
        - Changing concatenated key (pointer) field will cause integrity problem
        - Changing duplicate data fields will create inconsistency with DEDB

724

When accessing the secondary index with the ACCESS=INDEX parameter on the PCB, the calls that one may use are limited. One can use get calls although GNP and GHNP are not allowed since there are no children in a secondary index database. ISRT and DLET calls are not allowed. REPL calls are allowed but there are restrictions and warnings associated with REPL calls. REPL calls cannot change the search and subsequence fields. These are fields form the key of the index. REPL calls can change user data, concatenated key and duplicate data fields. Changing user data fields is OK since this is the way that their contents are managed. Users should not change the concaternated key field. Changing the concatenated key field will cause an integrity problem since this field contains the pointer to the target segment in the DEDB. Changing duplicate data fields will cause them to be inconsistent with the data in the DEDB.

REPL calls which attempt to change the search and subsequence fields and ISRT and DLET calls result in an ‘AD’ status code.

Accessing Secondary Index as Database

- When using

```
PCB TYPE=DB,DBDNAME=siname,…
```

  - GU, GHU, GN and GHN calls are allowed
    - GNP and GHNP are allowed but do not return data
  - ISRT calls are not allowed
  - DLET calls are allowed
    - They cause an inconsistency between the index and the DEDB
  - REPL calls are allowed
    - User data fields may be changed
    - Search, subsequence and concatenated key fields may not be changed
    - Duplicate data fields
      - May be changed when NOPROT is specified on the DBD
      - May not be changed when PROT is specified or defaulted on the DBD

When accessing the secondary index with a PCB which references the secondary index in the DBDNAME= parameter, different rules apply for which DLI calls may be used.

Any form of get call may be issued. Since there are no children in a secondary index database the GNP and GHNP calls will not return any segments.

ISRT calls are not allowed.

DLET calls are allowed, however, they will cause an inconsistency between the DEDB and the secondary index.

REPL calls are allowed. User data may be changed with a REPL call. Search, subsequence and concatenated key fields may not be changed. Duplicate data fields may be changed if NOPROT is specified on the ACCESS= parameter on the DBD. If PROT is specified duplicate data fields may not be changed. PROT is the default. The PROT and NOPROT specifications are explained on the next page.

Accessing Secondary Index as Database

- PROT|NOPROT subparameter on DBD ACCESS= parameter
  - Specified on DBD of secondary index

```
DBD NAME=siname,ACCESS=(INDEX,VSAM,PROT|NOPROT)
```

```
DBD NAME=siname,ACCESS=(INDEX,SHISAM,PROT|NOPROT)
```

    - PROT and NOPROT apply to access of secondary index with

```
PCB TYPE=DB,DBDNAME=siname,…
```

      - PROT
        - Disallows REPL calls which modify duplicate data
      - UNPROT
        - Allows REPL calls to modify duplicate data

726

The PROT or NOPROT subparameter on the ACCESS= parameter in the DBD applies to secondary indexes. It may be used with full function secondary indexes or with Fast Path secondary indexes. The syntax for Fast Path secondary indexes is shown here. For Fast Path secondary indexes this parameter only applies when the secondary index is specified on the DBDNAME= parameter on the PCB.

When the DBDNAME= parameter specifies a secondary index and NOPROT is specified on the DBD, REPL calls may change duplicate data in the secondary index. When PROT is specified or defaulted, REPL calls cannot change duplicate data. Since NOPROT allows duplicate data to be changed, it allows inconsistency to be created between the DEDB and the secondary index.

Accessing Secondary Index as Database

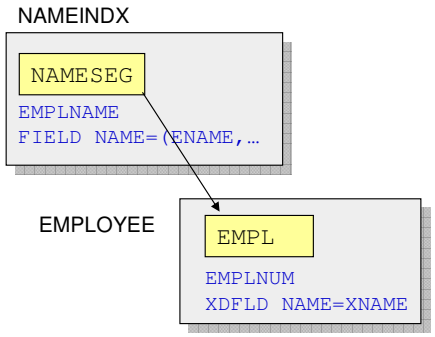
- Using ACCESS=INDEX
  - PSB
    - DEDB is specified on DBDNAME= parameter for PCB

```
PCB TYPE=DB,DBDNAME=EMPLOYEE,PROCSEQD=NAMEINDX,ACCESS=INDEX,…
```

    - SENSEG statements for DEDB segments from root to target must be specified
  - SSAs use secondary index segment and field names

```
GU NAMESEG (ENAME=SMITH)
```

  - DB PCB mask contains DEDB name in database name field, but secondary index segment name and secondary index key



727

When using ACCESS=INDEX on the PCB to access the secondary index as a database, the user application program differs from what it would be when using ACCESS=DB with the DBDNAME= parameter on the PCB specifying the secondary index.

When ACCESS=INDEX is used, the PCB must include one or more SENSEG statements for the DEDB segments. There must be a SENSEG statement for the DEDB root and each segment from the root to the target segment.

Even though the PCB SENSEG statements refer to the DEDB, if a call has an segment search argument (SSA), the SSA specifies the name of the index segment. Qualified calls use the names of fields in the secondary index segment.

The database PCB mask contains the DEDB name in the database name field, but the secondary index segment name in the segment name field and the key of the secondary index segment in the key feedback area.

Suppression of Index Maintenance

- Index maintenance may be suppressed for a BMP
  - User should create index after running the BMP
    - Index creation tool may be more efficient than index maintenance
- Index maintenance is suppressed with SETI control statement in DFSCTL data set
  - Suppression occurs for all DEDB secondary indexes for the BMP
  - Syntax:

```
//DFSCTL  DD *
SETI PSB=psbname
```


Users may suppress Fast Path secondary index maintenance by a BMP. Of course, this creates a mismatch between the DEDBs and their secondary indexes. After using index suppression, users should create or recreate the affected secondary indexes. This technique may be more efficient than using index maintenance with the BMP. This option may be attractive for users of a secondary index creation tool where the secondary indexes do not have to be in synch with the DEDBs during the execution of these BMPs.

Index maintenance suppression is invoked with a SETI control statement in the DFSCTL data set for a BMP. The SETI statement must specify the PSB used by the BMP. If the PSB= parameter is not specified or specifies a PSB not used by the BMP, the BMP abends with abend code U1060 and message DFS0510E is issued. The message text is:

```
DFS0510E THE SETI STATEMENT IS INVALID.  RGN n.
```

Fast Path DBs are not supported in DLI Batch processing, only BMP processing, in addition to online support.

Back Up and Recovery

- Standard HISAM and SHISAM back up and recovery are used
  - Image copies
  - Full function database change log records
  - Change Accumulation (optional without data sharing)
  - Database Recovery utility

729

Backup and recovery of Fast Path secondary indexes uses the standard IMS utilities and techniques for HISAM and SHISAM databases.

Since the secondary indexes are HISAM or SHISAM, database updates to them create standard full function log records. They may be accumulated with the Change Accumulation utility.

Reorganization

- Secondary Indexes cannot be reorganized with IMS utilities
  - Rebuilding these indexes with a tool is recommended
  - VSAM REPRO may be used as an alternative for SHISAM and HISAM with unique keys

Secondary indexes may become disorganized.

Fast Path secondary index HISAM databases cannot be reorganized with the HISAM Unload and HISAM Reload utilities or HD Unload and HD Reload utilities. They should be rebuilt to create reorganized indexes. Rebuilding the indexes requires a tool. It is assumed that all users of Fast Path secondary indexes will have a tool with the capability to build and rebuild secondary indexes.

If a HISAM secondary index has unique keys REPRO may be used to reorganize the database. It may not be used for HISAM secondary indexes with non-unique keys. Since segments are deleted by setting a flag in the delete byte when non-unique keys are used, REPRO is not appropriate for secondary indexes with non-unique keys. REPRO would not eliminate these flagged segments.

SHISAM databases are simply VSAM KSDSs without any special IMS fields such as segment codes or delete bytes. They may be reorganized with VSAM REPRO.

Command Reponses

- /DIS DB command response for Fast Path secondary index
  - TYPE of 'DHISNDX' indicates DEDB HISAM or SHISAM index

```
/DIS DB DEDBJN24 DEHSJX24
DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED ACC  CONDITIONS
DEDBJN24  DEDB    SEQ DEPEND DIRECT ADDRES UP   NOTOPEN
DB24A010  AREA    N/A    N/A    N/A    N/A      NOTOPEN
DB24A020  AREA    N/A    N/A    N/A    N/A      NOTOPEN
DEHSJX24  DHISNDX                          UP   NOTOPEN
*10299/134655*   IMS1
```

731

This page shows an example of the /DIS DB command response for a Fast Path secondary index. The display is done for a DEDB, DEDBJN24, and for its index, DEHSJX24. Under the 'TYPE' column 'DHISNDX' indicates that DEHSJX24 is a Fast Path secondary index.

Command Reponses

- QUERY DB command response for Fath Path secondary index
  - TYPE of 'DHISNDX' indicates DEDB HISAM or SHISAM index

```
QUERY DB NAME(DEHSJX24) SHOW(ACCTYPE)
DBName   AreaName MbrName   CC   TYPE          LAcc
DEHSJX24          IMS1       0   DHISNDX       UPD
```

  - QUERY DB NAME() with SHOW(SNDX) shows secondary indexes for the DEDBs

```
QUERY DB NAME(DEDBJN24) SHOW(SNDX)
DBName   AreaName SndxName MbrName   CC   TYPE
DEDBJN24                   IMS1       0   DEDB
DEDBJN24          DEHSJX24 IMS1       0   DHISNDX
DEDBJN24 DB24A010          IMS1       0   AREA
DEDBJN24 DB24A020          IMS1       0   AREA
```

732

This page shows two examples of QUERY DB commands with a Fast Path secondary index.

The first example is a query for a Fast Path secondary index. The response includes 'DHISNDX' in the type column.

The second example is a query with SHOW(SNDX). This is a request for the response to include the secondary index for the DEDB.

The QUERY DB NAME() SHOW(SNDX) command shows the associated Fast Path secondary index databases for a DEDB database. The SHOW(SNDX) option is mutually exclusive with other SHOW() options. The new "SndxName" output column shows Fast Path secondary index databases before DEDB Areas for a DEDB database. If a database specified in the NAME parameter is not a DEDB database and SHOW(SNDX) is specified, a completion code of '193' and completion text of 'NOT A DEDB' is returned for the DB resource. If a database is a DEDB database and it has no Fast Path secondary index database defined when SHOW(SNDX) is specified, a completion code of ' 194' and completion text of 'NO SECONDARY INDEX DEFINED' is returned for the DB resource.

Fast Path Secondary Indexes Summary

- Secondary indexes are HISAM or SHISAM databases
  - All pointers are symbolic (concatenated keys)
- Secondary indexes are maintained by IMS
  - Secondary index creation utility is not included in IMS
- Fast Path secondary indexes have unique capabilities
  - User data partitioning
  - Multiple secondary index segments
- PSBs using secondary indexes differ from full function
  - SENSEG statements are in physical sequence
  - PROCSEQD= is used to specify the secondary index
- Benefits
  - Alternate sequencing of data
  - Alternate key access to data
  - Access to data in secondary index without accessing the DEDB

733

Fast Path secondary index support provides full function databases, either HISAM or SHISAM, as secondary indexes for DEDBs. The pointers to the DEDB are always symbolic pointers. That is, they are the concatenated key of the target segment. This simplifies reorganization of DEDBs. Even though the target segments may be physically moved, the pointers do not have to be updated. IMS maintains these secondary indexes when the source segments are inserted, deleted, or replaced. IMS does not provide a utility to add a secondary index to an existing DEDB.

Fast Path secondary indexes have capabilities that are not included with secondary indexes for full function databases. These are user partitioning which allows the index to be spread across multiple HISAM ore SHISAM databases and multiple secondary index segments which allow multiple index entries to be created from different fields in the same source segment.

There are two significant differences in the implementation of secondary indexes from the implementation for full function databases. First, the SENSEG statements in PCBs must be in the physical sequence, not the secondary processing sequence. Second, the PROCSEQD= parameter, not the PROCSEQ= parameter, is used in the PCB.

Secondary indexing provides both alternate sequencing of segments in a database and access via an alternate key. When duplicate data is maintained in the secondary index, this data is available to applications without accessing the DEDB. This provides a performance benefit for some applications.