

# *Database*

**Information Management** software



## ***Database Enhancements***

- Dynamic Full Function Database Buffer Pools
- Miscellaneous Enhancements
- HALDB Enhancements

# Dynamic Full Function Database Buffer Pools

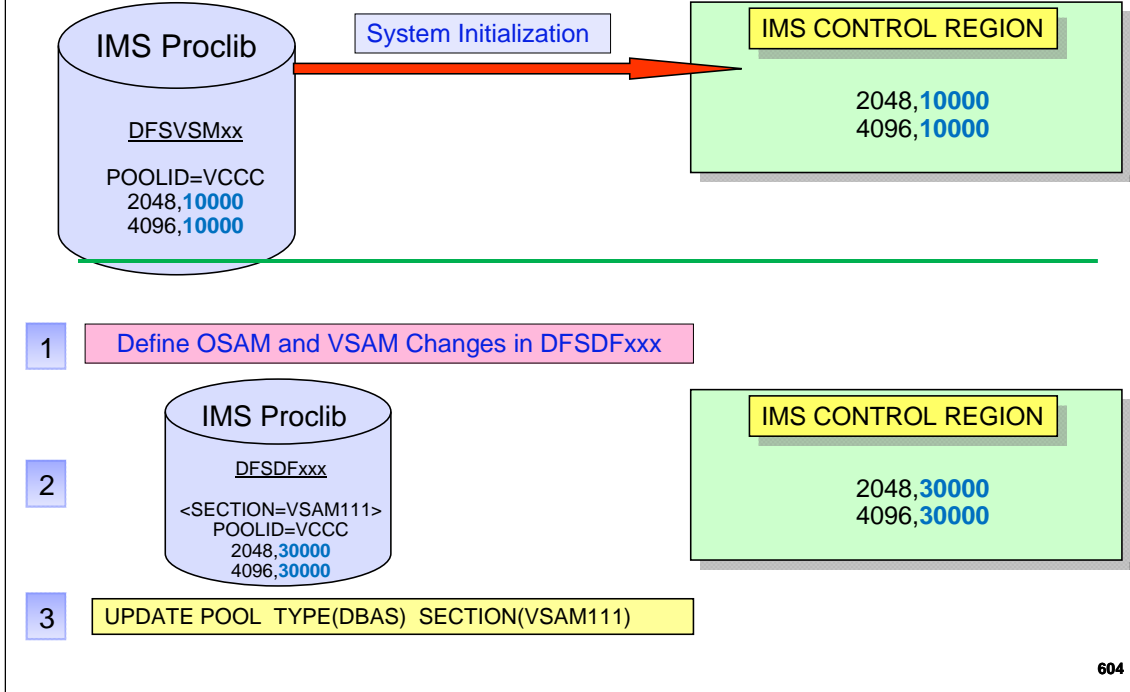
## ***Dynamic Full Function Database Buffer Pools***

- **IMS 12 adds dynamic buffer pool support for full function databases**
  - Users can dynamically manage full function buffer pools
  - Buffer pool definitions can change without taking IMS down
  - Application activity is internally quiesced
    - Allows new buffer pools to be created
    - Allows existing buffer pools to be changed or deleted
  
- **Benefits**
  - Improved buffer pool management
    - Provides type-2 commands for better usability
    - Eliminate system down time for modifications to buffer pool definitions
    - Improve application performance with improved buffer pool specifications

603

With IMS 12, users can add, change and delete full function buffer pools. This support is provided using new specifications in the DFSDFxxx proclib member in conjunction with the UPDATE POOL command. With this support, full function buffer pools can be managed without restarting IMS. IMS is able to internally quiesce application read and update activity to allow the UPDATE POOL command to complete with very little disruption to transaction workloads. Finally, with the ability to dynamically update full function buffer pool specifications, there can be better application performance when the buffer pools are sufficient to avoid unnecessary I/O.

## Overview of Dynamic Full Function Dynamic Buffer Pools



At system initialization, the full function buffer pool specifications are loaded from the DFSVSMxx member in the IMS PROCLIB data set. These buffer pool specifications can be changed dynamically by specifying new definition sections for OSAM and VSAM in the DFSDFxxx member of the IMS PROCLIB data set. An UPDATE POOL TYPE(DBAS) SECTION(OSAMxxx,VSAMxxx) must be issued to bring these full function buffer pool definitions into affect. These dynamic changes are retained across an Emergency Restart because they are stored in the Restart Data Set. However, the changes are lost with a subsequent Cold or Warm Start. To make the changes permanent, it is necessary to change the DFSVSMxx proclib member, too.

## ***Full Function Database Buffer Pools (Prior to IMS 12)***

- VSAM and OSAM definitions stored only in DFSVSMxx proclib member
  - IMS processes DFSVSMxx once during system initialization
  - No facility to change buffer pool definitions with online commands
  - Buffer pool modifications required an IMS system restart

In IMS 11 and earlier releases, the VSAM and OSAM buffer pool definitions were only stored in the DFSVSMxx proclib member. This member is only loaded once during IMS initialization. There is no facility to change the buffer pool definitions without changing the DFSVSMxx member in proclib and restarting IMS.

## VSAM Buffer Pools (as Defined in DFSVSMxx)

### ▪ VSAM Buffer Pool Specification

- POOLID=                      ←(One for each shared resource pool)
  - id,                              ←(shared resource pool user id)
  - FIXDATA=NO|YES,              ←(long-term page-fixing -- Data)
  - FIXINDEX=NO|YES,             ←(long-term page-fixing -- Index)
  - FIXBLOCK=NO|YES,            ←(long-term page-fixing – I/O Control Blocks)
  - STRINGNM=n                  ←(max VSAM I/O requests concurrently active)
- VSRBF=                      ←(One/more to define subpools within shared pool)
  - buffersize,                  ←(buffer size for subpool)
  - number of buffers,            ←(Number of buffers in this subpool)
  - type,                          ←(Index (I) subpool vs. Data (D) subpool)
  - HSO|HSR,                      ←(Specifies action if Hiperspace unavailable)
  - HSn                             ←(Number of hiperspace buffers for subpool)

606

In the DFSVSMxx proclib member, there are three parts to the VSAM buffer pool specification: 1) POOLID, 2) VSRBF, and 3) DBD. These are the keywords for the POOLID and VSRBF statements.

## VSAM Buffer Pools (as Defined in DFSVSMxx)

- VSAM Buffer Pool Specification (Continued)

– DBD=	←(Optionally assigned to POOLID shared pool id)
DBDname(	←(DBD from NAME= keyword on DBD macro)
dataset number,	←(Specific data set of data set group)
id,	←(Shared resource pool identifier)
ERASE=YES NO,	←(Treatment of deleted logical records)
FREESPACE=NO YES)	←(Treatment of defined free space % in KSDS)

- Example VSAM Specification in DFSVSMxx:

```
POOLID=VCCC
VSRBF=2048,8000,I
VSRBF=2048,26000,D
VSRBF=4096,32000,D
VSRBF=8192,104000,I
DBD=PVHDJ5B(B,VCCC,ERASE=YES,FREESPACE=YES)
```

These are the keywords for the DBD statement which allows a DBD to be assigned to a specific POOLID.



## OSAM Buffer Pools (as Defined in DFSVSMxx)

### ▪ OSAM Buffer Pool Specification

- IOBF=                              ←(One for each OSAM subpool definition)
  - length,                              ←(length of buffers in subpool)
  - number,                              ←(number of buffers in subpool)
  - fix1,                                ←(long-term page-fixing – buffers + prefixes)
  - fix2,                                ←(long-term page-fixing – prefixes + headers)
  - id,                                  ←(subpool identifier)
  - co                                  ←(caching option)
- DBD=                              ←(Optionally assigned to POOLID shared pool id)
  - DBDname(                            ←(DBD from NAME= keyword on DBD macro)
  - dataset number,                    ←(Specific data set of data set group)
  - id)                                 ←(Subpool identifier)

### ▪ Example OSAM Specification in DFSVSMxx

```
IOBF=(8192,8000,N,N,OCCC)  
DBD=POHIDKA(B,OCCC)
```

608

In the DFSVSMxx proclib member, the OSAM Buffer Pool specifications have two statements: 1) IOBF, and 2) DBD.

## Dynamic Full Function Database Buffer Pools

- **Dynamic VSAM and OSAM Buffer Pool Specification**
  - Initial definitions exist in DFSVSMxx proclib member
  - New and changed definitions are in the DFSDFxxx proclib member
    - OSAM format is similar to DFSVSMxx
    - VSAM format is similar to DFSVSMxx
      - POOLID in DFSDFxxx includes the VSRBF statement

```
Example:  
POOLID=  
(VSM1,  
FIXDATA=N,  
FIXINDEX=Y,  
FIXBLOCK=N,  
STRINGNM=255,  
VSRBF=(1024,30000,D),  
VSRBF=(1024,10000,I))
```

609

With IMS 12, there is a new feature for dynamically adding, updating and deleting VSAM and OSAM buffer pools. The initial VSAM and OSAM buffer pool specifications still exist in the DFSVSMxx proclib member and they are loaded during normal restart. However, new VSAM and OSAM buffer pools can be added and existing buffer pools can be changed using specifications in one or more DFSDFxxx proclib members in conjunction with the type-2 UPDATE POOL command.

## **Full Function Database Buffer Pools (DFSDFxxx)**

- Buffer Pool Specifications use <SECTION=section\_name>
  - <SECTION=OSAMxxx>
    - IOBF =(bufsize,bufnum,fix1,fix2,id,co)
    - DBD=(DBDname,dsid,id)
      - or, DBD(DBDname,dsid,id)
  
  - <SECTION=VSAMxxx>
    - POOLID=(id,FIXDATA=,FIXINDEX=,FIXBLOCK=,STRINGNM=,  
VSRBF=(buffersize,buffer number,type,HSO|HSR,HSn))
    - DBD=(DBDname,dataset number,id,ERASE=,FREESPACE=)
      - Or, DBD(DBDname,dataset number,id,ERASE=,FREESPACE=)

The OSAM buffer pool specifications in the DFSDFxxx proclib member are under Section headings. For example, the OSAM section is <SECTION=OSAMxxx> where xxx is any alphanumeric characters. Similarly, the VSAM buffer pool specifications are under the section heading <SECTION=VSAMxxx>.

## VSAM Buffer Pools (Changed Specification)

- VSAM Buffer Pool Specification Example

- Example VSAM Specification in DFSVSMxx:

```
POOLID=VCCC  
VSRBF=2048,8000,I  
VSRBF=2048,26000,D  
VSRBF=4096,32767,I,HSO,HS40000  
VSRBF=8192,32767,I,HSR,HS40000  
DBD=(PVHDJ5B(B,VCCC,ERASE=YES,FREESPACE=YES))
```

- Example of changed VSAM Specifications in DFSDFxxx

```
<SECTION=VSAMEXM>  
POOLID=(VCCC,  
    VSRBF=(2048,12000,I),  
    VSRBF=(2048,12000,D),  
    VSRBF=(4096,32767,I,HSO,HS80000),  
    VSRBF=(8192,32767,I,HSR,HS80000))  
DBD=(PVHDJ5B,B,VCCC,ERASE=YES,FREESPACE=YES)
```

611

This example shows the VSAM buffer pool specifications in DFSVSMxx and how they could be dynamically changed using buffer pool specifications in the DFSDFxxx proclib member in IMS 12 together with the UPDATE command.

## OSAM Buffer Pool (Changed Specification)

- OSAM Buffer Pool Specification Example

- Example OSAM Specification in DFSVSMxx:

```
IOBF=(8192,8000,N,N,OCCC)  
DBD=POHIDKA(B,OCCC)
```

- Example of changed OSAM Specification in DFSDFxxx

```
<SECTION=OSAMEXM>  
IOBF=(8192,12000,N,N,OCCC)  
DBD=(POHIDKA,B,OCCC)
```

This example shows the OSAM buffer pool specifications in DFSVSMxx and how they could be dynamically changed using buffer pool specifications in the DFSDFxxx proclib member in IMS 12 together with the UPDATE command.

## DFSDFxxx Considerations

- Multiple DFSDFxxx proclib members may be used
  - DFSDFMON
  - DFSDFTUE
- DFSDFxxx may have multiple section definitions

```

<SECTION=OSAMMON>
IOBF=( 8192,8000,N,N,OAAA)
DBD=( POHIDKA,B,OAAA)
<SECTION=VSAMMON>
POOLID=(VAAA,
        VSRBF=( 4096,32000,I,HSO,HS10))
DBD=( PVHDJ5B,B,VAAA,ERASE=YES,FREESPACE=YES)
<SECTION=OSAMTUE>
IOBF=( 8192,104000,N,N,OBBB)
DBD=( POHIDKA,B,OBBB)
<SECTION=VSAMTUE>
POOLID=(VBBB,
        VSRBF=( 4096,10320,I,HSO,HS10))
DBD=( PVHDJ5B,B,VBBB,ERASE=YES,FREESPACE=YES)

```

613

The VSAM and OSAM buffer pool specifications can be placed into different DFSDFxxx members in proclib since the UPDATE POOL command allows the user to specify the MEMBER keyword identifying the suffix of the DFSDFxxx proclib member in the proclib data set. Alternatively, the user can specify multiple VSAM and OSAM sections within one or more DFSDFxxx members.

## UPDATE POOL Command Support

### ■ UPDATE POOL Commands used to Add and Change buffer pools

#### – Add and Change Commands:

- Add or Change VSAM or OSAM buffer pool definitions:
  - UPDATE POOL TYPE(DBAS) SECTION(OSAMxxx)
  - UPDATE POOL TYPE(DBAS) SECTION(VSAMxxx)
- Or in one command:
  - UPDATE POOL TYPE(DBAS) SECTION(OSAMxxx,VSAMxxx)
- Add or Change definitions in an alternate DFSDFyyy proclib member
  - UPDATE POOL TYPE(DBAS) SECTION(OSAMxxx) MEMBER(yyy)

#### – Example: Add (8) 8K OSAM buffers, (24) 8K VSAM buffers

##### – DFSDFxxx

```
<SECTION=OSAMMON>
IOBF=(8192,8000,N,N,OAAA)
<SECTION=VSAMMON>
POOLID=(VAAA,VS RBF=(8192,24000,I))
```

##### – Issue: UPDATE POOL TYPE(DBAS) SECTION(OSAMMON,VSAMMON)

614

The ability to add or change VSAM and OSAM buffer pools requires both the DFSDFxxx proclib member specifications along with the type-2 UPDATE POOL command identifying the statement sections. The UPDATE POOL command can be issued individually for specific VSAM and OSAM sections, or the command can be issued for both VSAM and OSAM sections in the same command. The UPDATE POOL command can also reference a specific DFSDFxxx proclib member in the proclib data set using the MEMBER(yyy) keyword. In this case, yyy is the suffix used in DFSDFyyy. The default for yyy is 000.

## Deleting VSAM Buffer Pools

- UPDATE POOL Commands required to Delete subpools

- Deleting VSAM Subpool:

- Specify a POOLID=(*bufnum*) statement with a *bufnum* parm of 0
- UPDATE POOL TYPE(DBAS) SECTION() activates subpool deletion

- Example: Delete VSAM 4K subpool

- DFSDF<sub>xxx</sub>

```
<SECTION=VSAMTUE>  
POOLID=(VBBB,  
        VSRBF=(4096,0)  
        VSRBF=(8192,20000))
```

- Command issued:

- UPDATE POOL TYPE(DBAS) SECTION(VSAMTUE)

It is possible to delete a VSAM buffer pool by specifying a POOLID in a VSAM section with the VSRBF statement for the size of the buffer and a “0” for the number of buffers. The UPDATE POOL command is needed to complete the deletion of the VSAM buffer pool.

The database data set association with a subpool is established when the database data set is opened. If there is a database data set using a subpool that is to be deleted, the UPDATE POOL command must wait until the access to the subpool is completed before it can delete the subpool.

When the subpool is deleted, there is no association between the subpool and the database data set and the database data set can be associated with a new subpool by simply creating a new DBD= statement.



## Deleting OSAM Buffer Pools

- UPDATE POOL Commands required to Delete buffer pools

- Deleting OSAM Buffer Pool:

- Specify a IOBF=(*bufnum*) statement with a *bufnum* parm of 0
- UPDATE POOL TYPE(DBAS) SECTION() activates subpool deletion

- Example: Delete OSAM 4K subpool

- DFSDF<sub>xxx</sub>

```
<SECTION=OSAMTUE>  
IOBF=(4096,0,N,N,OCCC)
```

- Command issued:

- UPDATE POOL TYPE(DBAS) SECTION(OSAMTUE)

It is possible to delete an OSAM buffer pool by specifying the IOBF statement in the OSAM section using “0” for the number of buffers. The UPDATE POOL command is needed to complete the deletion of the OSAM buffer pool.

As with VSAM, the database data set association with a subpool is established when the database data set is opened. If there is a database data set using a subpool that is to be deleted, the UPDATE POOL command must wait until the access to the subpool is completed before it can delete the subpool.

When the subpool is deleted, there is no association between the subpool and the database data set and the database data set can be associated with a new subpool by simply creating a new DBD= statement.

## Querying VSAM and OSAM Buffer Pools

- **QUERY Commands required to Query buffer pools**

- **QUERY**

- POOL**
  - TYPE(DBAS)**
  - SUBTYPE(OSAM,VSAM) SIZE() POOLID()**
  - SHOW(ALL/STATISTICS/MEMBER)**

- **SHOW:**

- **STATISTICS**
    - Shows statistical information similar to /DIS POOL DBAS
  - **MEMBER**
    - Shows active proclib members used for buffer pool definitions
  - **ALL**
    - Shows both **STATISTICS** and **MEMBER** information

617

The QUERY POOL command is used to query information about the new and changed VSAM and OSAM buffer pools. The user can specifically limit the output to: 1) OSAM or VSAM buffer pools, 2) buffers of a particular size, or 3) specific pool ids. The options for the SHOW allows the user to show only statistical information that is similar to the current /DIS POOL DBAS command. Alternatively, the user can show the proclib member information used to add or update a buffer pool specification. It is also possible to show both statistical and member information using the ALL parameter.

## Querying VSAM and OSAM Buffer Pools (Example)

- QUERY POOL Commands Query Example 1:
  - QRY POOL TYPE(DBAS) SUBTYPE(OSAM,VSAM)

Response for: QRY POOL TYPE(DBAS) SUBTYPE(OSAM,VSAM)

Subpool	MbrName	CC	BufSize	PoolId	NBuf	ProcMbr	Section	FixOpt
OSAM	IMS1	0	512		10000	DFSDFGS1	OSAM001	N/N
OSAM	IMS1	0	1024	OSM1	16000	DFSDFGS1	OSAM001	N/N
VSAM-D	IMS1	0	1024	VSM1	20000	DFSVMGS		N/Y/N
VSAM-I	IMS1	0	512	VSM1	30000	DFSDFGS1	VSAM001	N/N/N

- Note: Some columns not shown

618

This example shows both VSAM and OSAM buffer pool specifications. It shows the proclib members used to create the various buffer pools and the VSAM and OSAM sections within each proclib member. There were other columns in the output of this command that are not shown here. There are: 1) LctReq/Rrba, 2) NewBlk/Rkey, 3) AltReq/BfAlt, 4) PurgRq/Nrec, 5) Fndlpl/SyncPt, 6) BfSrch/VRds, 7) RdReq/Found, 8) BfStlW/VWts, 9) PurgWr/HSR-S, 10) WBSyld/HSW-S, 11) WBSyWr/HSNBuf, 12) WRlseO/HS-W-F, and 13) NumErrors.

It should be noted that the /DIS POOL DBAS command will also show the dynamically added buffer pools.

## Querying VSAM and OSAM Buffer Pools (Example)

- Type-2 Commands Query Example 2:
  - QRY POOL TYPE(DBAS) POOLID(VSM1) SIZE(1024)

```
Response for: QRY POOL TYPE(DBAS) POOLID(VSM1) SIZE(1024)
```

Subpool	MbrName	CC	BufSize	PoolId	NBuf	ProcMbr	Section	FixOpt
VSAM-D	IMS1	0	1024	VSM1	8000	DFSVMGS		N/N/N

- Note: Some columns not shown

This example shows how the QUERY POOL command can be limited to a specific POOLID and a specific buffer pool size. There were other columns in the output of this command that are not shown here. There are: 1) LctReq/Rrba, 2) NewBlk/Rkey, 3) AltReq/BfAlt, 4) PurgRq/Nrec, 5) Fndlpl/SyncPt, 6) BfSrch/VRds, 7) RdReq/Found, 8) PurgWr/HSR-S, 9) WBSyld/HSW-S, 10) WBSyWr/HSNBuf, 11) WBSyRd/HS-R-F, 12) WRlseO/HS-W-F, and 13) NumErrors.

## ***UPD POOL TYPE(DBAS) Command Execution***

- **UPD POOL TYPE(DBAS) can *not* be Cancelled or Aborted**
  - Two execution possibilities:
    - 1) UPD command completes before SPOC Timeout
      - Reason codes (displayed in SPOC) show results of command changes
    - 2) UPD command completes after SPOC Timeout
      - Use QRY POOL TYPE(DBAS) commands to determine OR....
      - Use OM Audit Trail to determine changes made
        - Token “rqsttkn1” ties issued commands to command responses
  - Command may produce unintended or partial results

The UPD POOL TYPE(DBAS) command can not be cancelled or aborted once it is issued. There are two execution possibilities for this command. It can complete before the TSO SPOC timeout occurs or it can complete after the TSO SPOC timeout occurs. When the TSO SPOC timeout has occurred, the UPD POOL TYPE(DBAS) command continues to run in the background. If the UPD POOL TYPE(DBAS) command completes prior to the TSO SPOC timeout, then the results are shown on the TSO SPOC with reason codes next to each requested change. If the command completes after the TSO SPOC timeout, a series of targetted QRY POOL TYPE(DBAS) commands can be issued to determine the success or failure of the requested changes. It is also possible to use the OM Audit Trail to determine which changes succeeded, which changes failed, and which changes succeeded partially. The token “rqsttkn1” can be used to tie the commands in the OM Audit Trail to the command responses. The next several charts explain some issues with this command and how the command result might have unintended or partial results.

## ***Effect of Long-Running BMPS on UPD POOL TYPE(DBAS) Command***

- Long-running BMP can prevent UPD POOL TYPE(DBAS) completion
  - Two execution possibilities:
    - 1) Subpool is eventually freed up for change:
      - OSAM waits for target subpool to be “Not Owned”
        - See OSAM Buffer Pool Quiesce Processing
      - VSAM waits for DL/I activity to reach commit points
        - Note: All PSTs with affected subpools held until UPD completes
        - Advised to spread databases across multiple subpools
        - See VSAM Buffer Pool Quiesce Processing
    - 2) Subpools are not “freed up” for change
      - UPD command waits indefinitely

621

A long-running BMP can affect the ability of an UPD POOL TYPE(DBAS) command to complete since it can not be cancelled or aborted. There are two execution possibilities: 1) The subpool is eventually freed up by the BMP and the command can complete, and 2) the subpool is not “freed up” by the BMP and the UPD POOL TYPE(DBAS) command waits indefinitely for the subpool to become available for the change. OSAM and VSAM have different mechanisms for “freeing up” the target subpool and these are described in the more detail by the charts entitled “OSAM Buffer Pool Quiesce Processing” and “VSAM Buffer Pool Quiesce Processing” respectively. At a high level, OSAM waits for the target subpool to become “unowned”, while VSAM waits for DL/I activity for the target subpool to reach commit points. It should be noted that since there is no timeout for this command, all PSTs with subpools affected by this command will wait until the UPD POOL TYPE(DBAS) command completes. Spreading database data sets across many subpools will tend to lessen the impact of a change to any one of these subpools.

## ***UPD POOL TYPE(DBAS) can Have Partial Success***

- **UPD POOL TYPE(DBAS) change requests are processed Serially**
  - UPD command can have partial success and partial failures
  - Failed changes can be located anywhere in update sections
  - Determining partial success and failure of command:
    - Two possibilities:
      - 1) Failure occurred prior to SPOC Timeout
      - 2) Failure occurred after SPOC Timeout

There can be multiple change requests in a OSAM or VSAM section. The UPD POOL TYPE(DBAS) command processes the change requests serially. When the UPD POOL TYPE(DBAS) command is completed, the results may be total success, partial success, or total failure. There are two ways to determine whether an UPD POOL TYPE(DBAS) command worked and they depend on whether the command completed before or after the TSO SPOC timeout.

## ***UPD POOL TYPE(DBAS) can Have Partial Success***

- Failures occurred **Prior** to SPOC Timeout
  - Reason codes are displayed on SPOC

CC	CCText	Code
x'EA'	Dynamic buffer pool failure	N/A
x'EB'	Resource unable to be quiesced	N/A
x'EC'	Reduced buffer allocation	ICC_REDBUF
x'ED'	Minimum buffer allocation	ICC_MINBUF
x'1C0'	Poolid error in DFSDFxxx member	ICC_POOLID
x'1C1'	DBD error in DFSDFxxx member	ICC_INVDBD
x'1C2'	Lock request failed	ICC_LOCKFAIL

623

When the UPD POOL TYPE(DBAS) command completes before the SPOC timeout, then reason codes are displayed next to each change indicating whether the change request was completely or partially successful or whether the change request failed.



## ***UPD POOL TYPE(DBAS) can Have Partial Success***

- Failures occurred **After** SPOC Timeout
  - SPOC no longer available to show reason codes
  - Use targetted QRY POOL TYPE(DBAS) commands
    - Determine which changes were made partially or completely

When the UPD POOL TYPE(DBAS) command completes after the SPOC timeout, then the TSO SPOC is no longer available to display the reason codes for each change requested by the command. In this event, a series of targetted QRY POOL TYPE(DBAS) commands can be issued to show how each change request was performed.

## ***Satisfying UPD POOL TYPE(DBAS) Command if Storage is Unavailable***

- **When UPD POOL TYPE(DBAS) can not satisfy storage request**
  - IMS attempts to complete command with “less storage” than requested
    - OSAM
      - Attempts to allocate a smaller amount of storage
        - First attempt is 90% of requested amount
        - Second attempt is another smaller percentage
        - Threshold is reached at 4 buffers
    - VSAM
      - Attempts to allocate a smaller amount of storage
        - First attempt is 50% of requested amount
        - Second attempt is another smaller percentage
        - Threshold is reached at 3 buffers
  - Reason codes indicate requested storage was reduced
    - x'EC' = ICC\_REDBUF = Reduced buffer allocation

625

There are circumstances where the storage requested by the UPD POOL TYPE(DBAS) command can not be satisfied. Since IMS can not cancel or abort the command, it attempts to find the next best “working solution” to satisfy the UPD POOL TYPE(DBAS) command which means less storage for each requested subpool change. For OSAM, IMS will attempt to find a smaller amount of storage. The first attempt will be for 90% of the requested amount. If this is still unavailable, IMS continually tries to find a slightly smaller amount until the threshold of 4 buffers is reached. For VSAM, IMS will also try to find a smaller amount of storage to satisfy the change requested by the UPD POOL TYPE(DBAS) command. The first attempt will be for 50% of the requested amount of storage. The second attempt will be for a slightly smaller amount until the threshold of 3 buffers is reached. For both OSAM and VSAM, the x'EC' reason code will be displayed indicating that a reduced buffer allocation was needed to satisfy the UPD POOL TYPE(DBAS) command.

## ***VSAM Buffer Pool Quiesce Processing***

- **Activity against affected subpools is quiesced, subpool is destroyed**
  - IMS looks at PSTs to find PSBs with intent (read or update) on databases
    - If intent found, PST is quiesced at commit point
  - When all PSTs are quiesced
    - Buffer pools are purged
    - Open database data sets are closed and reopened
    - New applications with sensitivity to database
      - Held until UPDATE POOL command completes
      - Applies to both READ and UPDATE access
    - Old subpool is deleted and new subpool is created
  - Assigning a database data set to a new pool
    - Will not trigger quiesce processing
  - No TIMEOUT parameter for UPDATE POOL command
    - No ability to interrupt the command once issued

626

For VSAM, when the UPDATE POOL command is issued to add, change or delete one or more VSAM buffer pools, IMS must internally quiesce application activity against the affected buffer pools. The UPDATE POOL command must wait until all update activity has been committed and hardened to the database data sets. During quiesce processing, IMS looks at each PST to determine if it has read or update intent on a database affected by the UPDATE POOL command. If intent is found, the PST is quiesced at the next commit point. When all of the PSTs are quiesced, the buffer pools are purged to DASD. All open database data sets affected by the UPDATE POOL command are closed and reopened. When the UPDATE POOL command is in process, new applications with either READ or UPDATE intent are forced to wait until the command completes before they can use the desired VSAM buffer pool. Once the UPDATE POOL command has control of the affected buffer pool, it destroys the old buffer pool and creates the new buffer pool using the new specifications. If the only change is to assign a database data set to a new buffer pool, IMS will not perform quiesce processing. Unlike the DB QUIESCE command, there is no TIMEOUT parameter for the UPDATE POOL command. Once the command is issued, there is no way to interrupt it until it completes.

## ***OSAM Buffer Pool Quiesce Processing***

- **When subpool ownership goes to zero, subpool is destroyed**
  - Applications can own one OSAM (or VSAM) buffer at a time
  - UPDATE POOL command quiesces buffers
    - Only after application ownership of buffer goes to zero
  - After buffer is quiesced
    - Applications must wait for buffer to be reconfigured
  - Buffer reconfiguration causes:
    - Subpool to be purged and altered buffers to be committed
    - Subpool and buffer prefixes are released
    - Fixed pages and CF resources are released
    - Local cache is released
    - Sequential Buffers are invalidated
  - After reconfiguration of buffers
    - Applications requesting buffer resume processing

627

For OSAM, when the UPDATE POOL command is issued to add, change or delete one or more OSAM buffer pools, IMS must wait for the use of the affected buffers by applications to go down to zero. Before the usage count is zero, applications can use the buffers affected by the UPDATE POOL command. Once the usage count goes to zero, the buffer can be reconfigured. The reconfiguration process starts by purging the subpools causing any altered buffers to be committed. This causes the subpools, the buffer prefixes, the fixed pages, the CF resources and the local cache to be release. It also causes the Sequential Buffers to be invalidated. After the reconfiguration of the buffers is complete, applications may request the affected buffers and resume processing.

## **Buffer Pool Statistics and Database Data Set Reassignment**

- **Buffer Pool statistic handling differs for VSAM and OSAM**
  - VSAM statistics are reset
    - Old statistics are *not* carried over
  - OSAM statistics are accumulated
    - Old statistics are carried over
- **Database data set reassignment**
  - OSAM
    - Reassignment occurs after database data set is closed and reopened
    - Close of data set is explicit (not part of command)
  - VSAM
    - Reassignment occurs after database data set is closed and reopened
    - Close of data set is implicit (if target subpool is also changed)
      - If no change to target subpool, close of data set is explicit

628

The buffer pool statistics are handled differently for VSAM and OSAM following an UPDATE POOL command. For VSAM, the buffer pool statistics are reset and the old statistics are not carried over. It is advisable to do a QUERY POOL for the VSAM buffer pool statistics prior to issuing the UPDATE POOL command. The OSAM statistics are carried over and are not reset with the UPDATE POOL command.

When a database data set is reassigned from one buffer pool to a different buffer pool, the database data set must be closed and reopened. For OSAM, the closing and reopening of the database data set must be done explicitly. In other words, it is not performed as part of the UPDATE POOL command. For VSAM, the database data set must also be closed and reopened. However, if there is a corresponding change to the target buffer pool along with the reassignment of the database data set (ex. Increase in buffers), then the closing and opening of the database data set is done implicitly by the UPDATE POOL command.

## ***Initialization and IMS Restart for Buffer Pools***

- **Buffer Pool Initialization**
  - During IMS initialization, buffer pools are created using DFSVSMxx
  - Buffer pools are dynamically modified using DFSDFxx and UPDATE POOL
- **IMS Restart**
  - Committed buffer pool changes are written to Restart Data Set (RDS)
    - Emergency Restart will restore buffer pools using RDS
    - Normal Restart will initialize buffer pools from DFSVSMxx
  - XRF Takeover load committed changes from RDS
  - RSR and FDBR do not track committed changes
- **Log Records**
  - UPDATE command changes are logged with x'22' record
    - Log record is for information only
    - Non-recoverable command

629

As in IMS 11 and earlier versions, the buffer pools are initially created during IMS initialization using the buffer pool definitions in the DFSVSMxx proclib member. These specifications are also read during a normal restart of IMS. In IMS 12, full function buffer pools can be added or changed using VSAM and OSAM definitions specified in the DFSDFxxx proclib members in conjunction with the UPDATE POOL command identifying the sections containing the new and changed buffer pool specifications.

IMS stores the new and changed buffer pool specifications in the Restart Data Set (RDS) and during an emergency restart, IMS restores these definitions from the RDS. The RDS is also used to restore definitions during an XRF takeover. However, RSR and FDBR do not read the RDS and therefore can not restore the new and changed buffer pool specifications. FDBR has its own buffer pools and they are not affected by this new feature. If an UPDATE POOL command is issued for RSR or FDBR, it is ignored.

The UPDATE POOL command logs information in the x'22' log record for information purposes only. The UPDATE POOL command itself is non-recoverable.

## ***Dynamic Full Function Database Buffer Pools Summary***

- IMS 12 adds dynamic buffer pool support for full function databases
- Users can dynamically manage full function buffer pools
- Buffer pool definitions can change without taking IMS down
- For VSAM, application activity is internally quiesced at commit
- For OSAM, buffer activity is quiesced when application usage is zero
  
- **Benefits**
  - Improved buffer pool management
    - Provides ability to change specifications dynamically for better usability
    - Eliminate system down time for modifications to buffer pool definitions

630

With IMS 12, users can add, change and delete full function buffer pools. This support is provided using new specifications in the DFSDFxxx proclib member in conjunction with the UPDATE POOL command. With this support, full function buffer pools can be altered without restarting IMS. IMS is able to internally quiesce application update activity to allow the UPDATE POOL command to complete with very little disruption to transaction workloads.

## ***Miscellaneous Database Enhancements***

### ▪ **Miscellaneous Enhancements**

- Display status of randomizers and partition selection exit routines
- Retry after lock timeouts
- Improved information with lock timeouts
- Batch Data Sharing Abend Elimination
- Increased VSAM pools from 16 to 255
- CA Reclaim Support
- New command codes for sequential search
- CICS threadsafe support
- IRLM 2.3



## ***Miscellaneous Database Enhancements***

- **Miscellaneous Enhancements**
  - Reuse of Local DMB Numbers
  - Display status of randomizers and partition selection exit routines
  - Retry after lock timeouts
  - Improved information with lock timeouts
  - Batch Data Sharing Abend Elimination
  - RACF userid in Data Capture batch log records
  - Increased VSAM pools from 16 to 255
  - Temporary close of VSAM data sets when acquiring new extents
  - Elimination of OSAM U0080 Open/Close/EOV Abends
  - DFS993 sent to system console
  - CA Reclaim Support
  - New command codes for sequential search
  - CICS threadsafe support
  - IRLM 2.3

## ***Reuse of Local DMB Numbers***

- **IMS creates controls blocks for each database in the system**
  - Created by DATABASE macro or CREATE DATABASE command
  - There is a limit of 32,767 of these control blocks
  
- **IMS 12 allows these numbers to be reused**
  - After a database definition has been deleted by DRD or Online Change
    - Previous versions did not reuse the numbers
  
- **Benefit**
  - Cold start of IMS is not required when these numbers reach 32,767

633

The local database number (DMB number) is an internal value assigned by IMS when a database is defined to an IMS online system. Prior to IMS 12, this local database number is never reused when its database is deleted by online change or DRD. A cold start is required when the local database number reaches the limit of 32K-1. In IMS 12 the local database numbers deleted by online change or DRD can be reused when databases are added by online change or DRD.

This change only affects local DMB numbers. Global DMB numbers are reused in previous versions of IMS. A global DMB number is assigned to a database when it is registered with DBRC. The reuse of global DMB numbers was introduced in IMS 9.

## Status Messages for DB Exit Routines

- Status message issued for randomizer when (P)HDAM database is opened by command

```
DFS2842I RANDOMIZER name FOR database IS LOADED | SHARED
```

- 'LOADED' appears when routine is loaded from library
- 'SHARED' appears when routine is already resident due to use by another database

- Status message issued for randomizer when (P)HDAM database is closed by command

```
DFS2838I RANDOMIZER name FOR database IS DELETED AND GONE | SHARED
```

- 'GONE' appears when routine is deleted from memory
- 'SHARED' appears when routine remains in memory and used by another database

634

If an HDAM or PHDAM database is opened as the result of a command, message DFS2842I is issued. Either LOADED or SHARED appears in the message. LOADED appears when the routine is loaded as a result of the open of the database. SHARED appears when the routine is already in memory due to its use by another database. When the database is closed as the result of a command, either GONE or SHARED appears in the message. GONE appears when the routine is deleted from memory. SHARED appears when the routine remains in memory due to its use by another database.

The DFS2842I message is issued for full function databases as a result of the following commands:

```
/START DB dbname OPEN
UPDATE DB NAME(dbname) START(ACCESS) OPTION(OPEN)
```

The DFS2838I message is issued for full function databases as a result of the following commands:

```
/DBR DB dbname
/DBD DB dbname
/STO DB dbname
/STA DB dbname
UPDATE DB NAME(dbname) STOP(ACCESS\UPDATES\SCHD)
UPDATE DB NAME(dbname) START(ACCESS)
```

The *dbname* in these commands may be a HALDB partition name.

## Status Messages for DB Exit Routines

- Status message issued for partition selection exit routine when HALDB database is opened or closed by a command

```
DFS2406I THE HALDB PARTITION SELECTION EXIT ROUTINE rname
FOR THE HALDB dbname IS LOADED|GONE|SHARED
```

- 'LOADED' appears when routine is loaded from library
  - 'GONE' appears when the routine is deleted from memory
  - 'SHARED' appears when routine is already resident or remains in memory due to use by another database
- Benefit
    - Allows users to easily determine that an exit routine has been unloaded or a new one has been loaded when replacing the exit routine

635

If a HALDB database uses a partition selection exit routine the DFS2406I message is issued when the database is opened or closed as the result of a command. When the database is opened, either LOADED or SHARED appears in the message. LOADED appears when the routine is loaded as a result of the open of the database. SHARED appears when the routine is already in memory due to its use by another database. When the database is closed, either GONE or SHARED appears in the message. GONE appears when the routine is deleted from memory. SHARED appears when the routine remains in memory due to its use by another database.

Commands which might cause the DFS2406I message to be issued include:

```
/START DB HALDBmaster OPEN
UPDATE DB NAME(HALDBmaster) START(ACCESS) OPTION(OPEN)
UPDATE DB NAME(HALDBmaster) STOP(ACCESS|UPDATES|SCHD)
/DBR DB HALDBmaster
/DBD DB HALDBmaster
```

These messages are especially useful when replacing a shared exit routine. They clearly indicate if the old routine has been deleted and if a new routine has been loaded.

## ***Retry After Lock Timeouts***

- **Transaction is retried after a U3310 lock timeout abend**
  - Exceptions for IFP, CPI-C driven programs, and protected conversations
    - These messages are discarded
  - Retry can be overridden by DFSNDMX0 exit routine
  - Previous IMS versions always discarded input message after a timeout
    - Unless DFSNDMX0 indicated that a retry should be done
  
- **Benefit**
  - Default is to attempt to process the input message

636

Lock timeouts occur when the IRLM is used as the lock manager and the IMS LOCKTIME value is exceeded. Installations may choose whether lock timeouts produce a 'BD' status code for the DL/I call or a U3310 abend. Previous versions of IMS discarded the input message if a transaction was abended due to a lock timeout. In this case, there was no retry of the transaction. The Non-Discardable Messages exit routine (DFSNDMX0) may be used to override the discard decision. There is no retry for IFP regions, CPI-C driven applications, and protected conversations. IFP region messages are not retried for performance reasons. CPI-C driven applications receive messages directly from APPC-MVS and IMS does not manage the messages. The RRMS context token is no longer valid after the abend of an RRS protected conversation.

IMS 12 changes the default processing after a U3310 due to a lock timeout. With IMS 12 the input message is not discarded and the transaction is retried. This may be changed by the Non-Discardable Messages exit routine (DFSNDMX0).

## ***Lock Timeout Message and Logging***

- **IMS 12 adds optional DFS2291I diagnostic messages for lock timeouts**
  - Timeouts occur only with IRLM and IMS LOCKTIME specified
  - Previous IMS releases provide information only via RMF reports
  
- **IMS 12 writes log record x'67D0' subtype x'1B' for lock timeouts**
  - Contains same information as the DFS2291I message
  
- **Benefit**
  - Information on lock conflicts is more readily accessible

637

The RMF II ILOCK (IRLM Long Lock Detection) Report includes information about the waiters and blockers when a lock request exceeds the IRLM TIMEOUT value. If the wait for a lock exceeds the IMS LOCKTIME value when using the IRLM, the waiter is abended with a U3310 or a 'BD' status code is returned to the program. The U3310 or 'BD' is determined by the "STATUS" or "ABEND" specification on the LOCKTIME specification in IMS. IMS 12 adds an IMS message to provide more readily available diagnostic information.

Long lock timeouts cause IMS to write a x'67D0' subtype x'1B' log record. This log record contains the same information that is included in the DFS2291I message. This message is documented on the next page.

## Lock Timeout Message

- New DFS2291I message issued with U3310 abend or 'BD' status code
  - U3310 or 'BD' indicates that waiter has exceeded the specified wait time
  - DFS2291I is either a multiple line message

```
DFS2291I LOCKNAME=0900004288800201D7
DFS2291I DBNAME=DLVNTZ02 LOCKFUNC=GET LCL AND GBL ROOT LOCKS
DFS2291I BLOCKER PST=0001 TRAN=NQF1 PSB=PMVAPZ12 TYPE=MPP
DFS2291I BLOCKER TRANELAPSEDTIME=00:01:11 IMSID=IMS1
DFS2291I BLOCKER RECOVERY TOKEN=IMS1 0000000200000000
DFS2291I VICTIM PST=0002 TRAN=SHF1 PSB=PMVAPZ13 TYPE=MPP
DFS2291I VICTIM TRANELAPSEDTIME=00:00:49 IMSID=IMS1
DFS2291I VICTIM RECOVERY TOKEN=IMS1 0000003000000000
```

- Or a "short" one line message

```
DFS2291I BLOCKER PST=0001 TRAN=NQF1 PSB=PMVAPZ12 TYPE=MPP
```

638

This shows examples of the DFS2291I message. The first example is for the multiple line message. If there are other waiters for the same lock, they are also listed with the word "WAITER" where "VICTIM" appears in this example. The second example is for the single line or "short" message. The IMSID= field is added by IMS 12 APAR PM30851. For batch jobs the IMSID value is blanks.

In this example transaction NQF1 using PSB PMVAPZ12 holds a local and global root lock in database DLVNTZ02. This transaction's elapsed time is now 1 minute and 11 seconds. Transaction SHF1 using PSB PMVAPZ13 is waiting on this lock. Its elapsed time is now 49 seconds.

## Lock Timeout Message

- Installation chooses whether the DFS2291I messages are issued
  - Parameter in DIAGNOSTIC section of DFSDFxxx

```
<SECTION=DIAGNOSTIC>
MSG2291I=ISSUE | SHORT | SUPPRESS
```
  - SUPPRESS is the default
  - ISSUE creates multiline messages
  - SHORT creates one line messages
  
- Number of messages is limited for one U3310 situation
  - Message is issued only for the first five U3310s for a transaction

639

The DFS2291I messages are only issued if they are requested by specifying MSG2291I=ISSUE or MSG2291I=SHORT in the DIAGNOSTIC section of the DFSDFxxx member. ISSUE causes multiple line messages to be issued. SHORT causes one line messages to be issued.

It is possible that the retry of a transaction after a timeout will result in another timeout. This could occur multiple times. The DFS2291I message will be issued only for the first five U3310 abends for an input message.



## **Batch Data Sharing Abend Elimination**

- **Batch Data Sharing jobs survive CF cache structure access failures**
  - Previous releases produced U3303 abends when access to OSAM or VSAM cache structures failed
  - IMS 12 causes batch data sharing job to wait for a resolution of the structure problem
    - Message issued:
      - DFS2404A AN ERROR WAS ENCOUNTERED WHEN ACCESSING THE COUPLING FACILITY. STRUCTURE xxxxxxxxxxxxxxxxxxx RSN yy
- **Benefit**
  - Improved availability and ease of use for batch data sharing jobs
  - Users may move and rebuild OSAM and VSAM structures while batch jobs are executing

640

In previous versions of IMS a batch data sharing job would abend with a U3303 when an OSAM or VSAM cache structure access failed. For example, an access attempt while a structure was being rebuilt would fail. This problem did not occur with online systems. They survived access failures. They waited for the resolution to the structure access problem. IMS 12 allows batch jobs to survive when these structure accesses fail. Like online systems, they wait for the resolution to the problem. When the problem is resolved, the batch jobs continue processing. For example, when a rebuild of a structure completes, the batch jobs continue.

If the batch job detects the failure, it issues the new DFS2404A message. The reason code in the message is used to identify the type of failure that occurred when the batch job attempted to access the structure.

This enhancement allows users to rebuild their OSAM and VSAM cache structures while their data sharing batch jobs are executing. This may be done to address coupling facility failures or to move structures between coupling facilities for reconfigurations. In previous versions of IMS, batch jobs did not survive these rebuilds.

This enhancement does not eliminate all U3303 abends for batch jobs. It only eliminates those caused by cache structure access failures.

## ***RACF userid in Data Capture batch log records***

- The RACF userid is added to the Data Capture log records for batch jobs
  - Log record x'9904'
  
- **Benefit**
  - Asynchronous changed data capture users have access to RACF userid in log records

641

In previous versions of IMS, the RACF userid only appears in changed data capture log records (type x'9904') when the log is produced by an online system. IMS 12 adds the RACF userid to these log records when they are produced by batch (DLI or DBB) jobs. Changed data capture writes log records when the LOG (asynchronous) option is chosen in the DBDGEN.

The RACF userid is specified by including the USER= parameter on the JOB statement of the batch job.

## ***Increased VSAM Pools***

- **IMS 12 allows up to 255 VSAM database buffer pools**
  - Previous versions were limited to 16 pools
- **Requires IMS 12 APAR PM28721**
- **Requires DFSMS APAR OA32318**
  - PTF UA57797 for z/OS V1R11 and PTF UA57798 for z/OS V1R12
- **Implementation**
  - Users may specify up to 255 POOLID statements in DFSVSMxx member or DFSVSAMP data set
- **Benefits**
  - More VSAM pools and subpools may be specified
    - Increases capabilities to tune VSAM pools for database performance

642

Previous versions of IMS allowed only 16 VSAM full function database buffer pools to be defined for an IMS online system, batch job, or utility. IMS 12 expands this to 255 for online systems and 254 for batch jobs and utilities. Each buffer pool may have separate subpools for different buffer sizes and for data and index components.

VSAM buffer pools are defined with POOLID statements in the DFSVSMxx member or DFSVSAMP data set. IMS 12 allows users to specify up to 255 of these POOLID statements.

The additional buffer pools give users more flexibility in tuning their systems for full function database performance.

## ***Temporary Close of VSAM Data Sets with New Extents***

- **IMS 12 issues a temporary close (CLOSE TYPE=T) when a new extent is taken on a VSAM database data set**
  - CLOSE TYPE=T updates the catalog but leaves the data set open
  - Jobs which open the database data set but do not do verifies will be aware of the new extent
  
- **Benefit**
  - Tools which read a database while it is open in online systems or batch jobs are aware of the new extents

643

APAR PM29094 for IMS 12 changes the actions of IMS when a new extent for a VSAM database data set is created. IMS issues a temporary close of the data set. This is a CLOSE with TYPE=T. The temporary close leaves the data set open but updates the catalog with the information about the new extent.

The temporary close is important for some tools. They may open the database data set without a verify. If the open is done while the data set is in use by an online system or batch job and the online system or batch job has extended the data set, the open by the tool previously would not be aware of the new extent. The temporary close makes the new extent information available to the tool. When the tool was unaware of the new extent, it might produce erroneous information about the database.

Unlike some tools, IMS batch jobs (DLI or DBB) and online systems do not need the temporary close. They always do a verify when opening a VSAM database data set. The verify makes the new extent information available to them.

## ***Elimination of OSAM U0080 Open/Close/EOV Abends***

- **IMS 12 eliminates IMS U0080 abends for OSAM Open, Close, and EOV processing**

- Previous versions caused IMS system abends
- IMS 12 closes the database and marks it 'recovery needed'
- IMS 12 issues message DFS0730I:

```
DFS0730I UNABLE TO OPEN|CLOSE DATASET WITH DDNAME ddname
FOR REASON X, yy, z DATABASE dbdname programid
```

- When data set cannot be extended, enhanced DFS0842I message is sent:

```
DFS0842I OSAM DATASET CANNOT BE EXTENDED, REASON=x, z dbdname
DFS0842I ddname, dsname
```

- **Benefit**

- Improved system availability
- Improved diagnostic information

644

In previous versions of IMS there were some problems in OSAM open, close or end-of-volume processing that could result in U0080 abends. These were rare. When they occurred, the entire IMS system terminated with the U0080 abend. IMS 12 has changed this processing. When such problems occur, the database is closed and marked 'recovery needed'. The abend does not occur. Additionally, message DFS0730I is issued for open or close problems. 'X' is included as the first character in the reason code. The 'yy' and 'z' values identify the actual problem. In previous versions, the DFS0730I message was not issued with the U0080 abend. This made it more difficult to determine the database and data set with the problem. When the data set cannot be extended as part of EOV processing, message DFS0842I is issued. This message is enhanced in IMS 12 to include a subcode (z) to further explain the reason for extension failure.

Even though IMS 12 eliminates abend U0080 for OSAM open, close and EOV processing, it may still be issued for OSAM entry and termination processing.

## ***Message DFS993I Sent to System Console***

- **Message DFS993I is sent to both the master terminal and the system console**

```
DFS993I (CSA PSB|DLS PSB|PSBW) POOL TOO SMALL, UNABLE TO  
SCHEDULE PSB PSBNAME
```

- Previous versions sent the message only to the Master Terminal
  - Some systems (e.g. DBCTL) do not have a Master Terminal
- **Benefit**
  - Message identifying the problem is readily available to all users

645

IMS 12 sends the DFS993I message to both the system console and the Master Terminal. Previous versions of IMS sent this message only to the Master Terminal. Since DBCTL systems do not have a Master Terminal, they did not receive the message.

The DFS993I message is issued when the PSB Work, CSA PSB or DLI PSB pool is too small. IMS 12 allows DBCTL users to easily determine why a PSB schedule failure occurs because of insufficient space in one of these pools.

## CA Reclaim Support

- **IMS may use CA reclaim support for KSDSs with z/OS 1.12**
  - z/OS 1.11 does not include a CA reclaim capability
    - CI reclaim does not reclaim the empty CI with the highest key in the CA
    - The index structure is maintained for these CAs with no records
  - z/OS 1.12 provides CA reclaim support
    - All CIs in a CA may be reclaimed
      - When all CIs in a CA are empty, the CA is reclaimed
      - The index structure is reduced by eliminating this CA
      - The CA may be reused for records with other keys

646

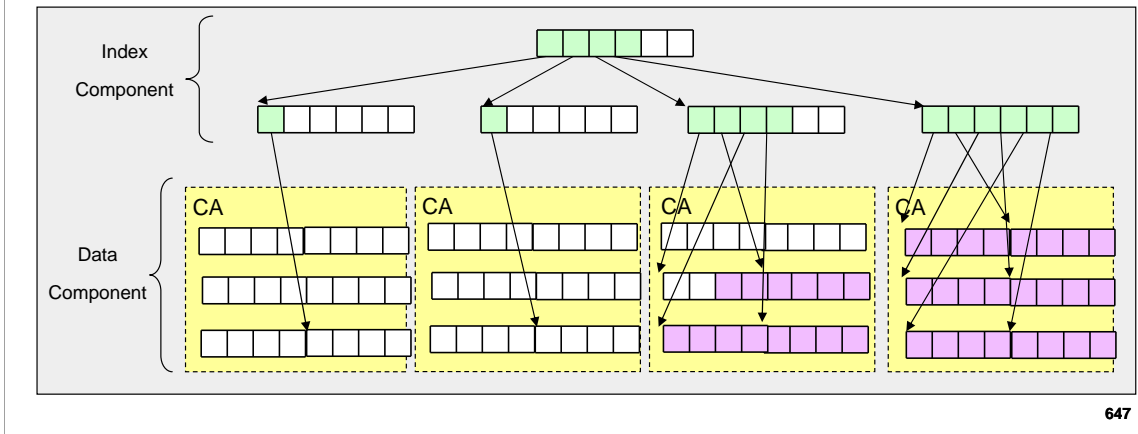
When IMS database KSDS records are erased with z/OS 1.11 and previous releases, VSAM CI reclaim does not reclaim the empty CI that has the highest key of the CA. This otherwise empty CA occupies the index structure as if it was not empty. If an application re-inserts records with the erased keys or keys of nearby values, those empty CAs are reused. However, if the application erases a range of keys and does not reuse those keys or only inserts records with ever higher keys, VSAM does not reclaim or reuse those empty CAs with lower keys. The failure to reclaim the CAs not only results in wasted disk space but also could cause performance problems in index search because much of the index structure could be populated with those empty index records.

The CA Reclaim feature in z/OS 1.12 allows free CA space to be reused. With CA Reclaim, space fragmentation caused by erasing records from a KSDS will be minimized to reduce the need to reorganize the data set. When the freed CAs are placed in a free chain to be reused, the index structure can be shrunk to facilitate quicker data accesses. When space is needed for a new CA, a CA from the free chain is reused so there will be fewer calls to EOVS to extend the KSDS.

There is no requirement for all of the systems in a sysplex to be at the same z/OS release level. z/OS 1.10 and z/OS 1.11 have compatibility maintenance so that they may process data sets for which CA reclaim is being used with z/OS 1.12. However, CA reclaim is only processed on systems that have z/OS 1.12.

## CA Reclaim Support

- The problem without CA reclaim
  - Typically occurs with increasing key values and deletion of old records
  - Empty CAs cannot be reused
  - Index points to empty CAs
    - Reorganization is required to use the empty space and optimize the index



647

This slide illustrates a problem that may occur when CA reclaim is not available. The problem typically occurs when new records have increasing key values and old records are deleted. The CAs which contained the old records become empty in the sense that they contain no records. CIs in the CA are reclaimed when all of their records are deleted with one exception. The reclaimed CIs are available for the insertion of records in the same CA. They are not available for use by another CA. The exception is that the CI with highest key in the CA is not reclaimed. The index entry pointing to this CI is maintained.

When many records with low valued keys are deleted, many CAs may be unused. Nevertheless, they cannot be reused. This may consume a lot of space as new records are inserted and old records are deleted. Even though the total number of records in the data set does not grow, the data set must grow. The index continues to point to these empty CAs. Sequential processing from the beginning of the data set may have to read many index entries before it finds an actual record.

Without CA reclaim the solution to this problem is to reorganize the data set. This requires a database outage with the exception of HALDB Online Reorganization for PHIDAM primary indexes. HALDB Online Reorganization allocates a new primary index.



## CA Reclaim Support

- **Specification of CA reclaim**
  - CA reclaim may be specified by data class
    - CA Reclaim is invoked for a data set when the data class has it specified
      - May be overridden for individual data sets with the ALTER command
  - There is no specification in IMS
- **CA Reclaim may be used with any version of IMS**
- **CA Reclaim statistics are available with LISTCAT output and in SMF type 64 records.**
  
- **Benefits**
  - Fewer reorganizations required
  - Improved disk space usage
  - Especially useful when new keys have increasing values

648

CA reclaim is invoked under z/OS 1.12 when the data set is defined with a data class for which CA reclaim is specified. No IMS external is required to exploit this function. It occurs automatically for all IMS versions when they execute under z/OS 1.12 or later and CA reclaim is specified for the data set.

The SYS1.PARMLIB IGDSMSxx member determines if CA reclaim may be used by a system. CA\_RECLAIM(NONE) is the default and disables CA reclaim in the system. CA\_RECLAIM(DATACLASS) allows CA Reclaim for data sets. When CA Reclaim is allowed for a system it is used for a KSDS if its data class has CA\_Reclaim(Y) specified when the KSDS is defined. CA\_Reclaim(Y) is the default for data classes. CA Reclaim is disabled for KSDSs when they are defined when CA\_Reclaim(N) is specified for the data class.

CA Reclaim may be enabled or disabled for individual data sets with the ALTER RECLAIMCA or ALTER NORECLAIMCA command. The ALTER command will take effect at the first OPEN following the CLOSE of all open ACBs for the data set.

CA Reclaim statistics are included with IDCAMS LISTCAT output in z/OS V1R12. The number of CAs reclaimed (REC-DELETED) and reused (REC-INSERTED) are in the INDEX component of a LISTCAT. Without CA Reclaim support, these numbers were always 0. CA Reclaim statistics are also available in SMF type 64 records.

The benefits of CA Reclaim include:

- Fewer reorganizations are required. They are not needed since CA reclaim is able to use the space from the CAs with deleted records for CAs with other keys.
- Disk space usage is improved. Data sets for which large ranges of keys are deleted do not have to grow to provide space for new records which can use the old space.

These benefits are especially useful for databases where new records with increasing key values are added while old records are deleted. They also apply to other data sets where all of the records in the key range for a CA are deleted.

## ***New Command Codes for Sequential Search***

- **IMS 11 SPE and IMS 12 provide two new command codes for GN and GHN calls for root segments**
  - A: clears positioning and causes call to start at the beginning of database
  - G: prevents randomization for HDAM, PHDAM, and DEDB calls and does not call the partition selection exit routine for PHDAM and PHIDAM.
    - Causes a sequential search of database when used with SSA which specifies a range of values for the root key
      - Successive calls do not return results in key sequence
- **Recommendation**
  - Do not use these commands codes with DL/I calls
    - Alternative explained on next page
- **Benefit**
  - Used by JDBC with Open Database to provide results comparable to relational

649

An SPE for IMS 11 adds two new command codes for use with GN and GHN calls for root segments. These new command codes are also included in IMS 12.

Command code A clears positioning and causes the GN or GHN call to start at the beginning of the database.

Command code G is used with GN and GHN calls qualified on the root segment key. It prevents the use of the randomizer routine for DEDB, HDAM, and PHDAM databases. It also prevents the use of the partition selection exit routine for a PHDAM or PHIDAM databases. Instead of using these routines, the call processes the next records in the database until one satisfying the SSAs is found or the end of the database is reached. This changes the actions that would otherwise occur for calls qualified on the key of the root segment. Consider a call qualified with key  $\geq 1000$  and key  $\leq 2000$ . Without the A or G command codes the call randomizes using key=1000. It returns the first segment found if it satisfies the SSAs. Otherwise, it returns a 'GE' (not found) status code. If a segment is returned, successive calls using the same SSAs would move forward in the database. If a segment not satisfying the call is encountered, a 'GE' status code is returned. On the other hand, the use of the A command code causes the call to begin at the beginning of the database. The G command code causes the call to ignore roots which do not satisfy the SSAs and continue to following roots until one is found that satisfies the SSAs.

If one uses these new command codes, one should use the A command code on the first GN or GHN call and not use it on the successive calls. If A were used on every call, the calls would always start at the beginning of the database and return the same segment.

These command codes should not be used with DL/I calls. There is a much more efficient alternative which is explained on the next page.

The new command codes are added to IMS to provide support for JDBC use with Open Database. JDBC users expect to get the same results with IMS and relational databases. The JDBC drivers use these command codes to provide the same results.

## ***New Command Codes for Sequential Search***

- **JDBC driver uses these command codes when converting call to DL/I**
  - Without these command codes a search on a range of keys does not return all values for DEDB, HDAM, or PHDAM databases or for PHDAM and PHIDAM databases with a PSE routine
    - Search begins with the low key value and ends when key greater than high key value is found
    - With these command codes the search is for the entire database
    - Call returns expected results
      - With a performance cost
- **A better alternative**
  - Create a secondary index on this key
  - Use PCB with PROCSEQ=secondaryindex for full function databases
  - Use PCB with PROCSEQD=secondaryindex for DEDBs
  - Avoids sequential scan of database

650

The JDBC driver uses the new command codes when converting a JDBC call to a DL/I call. This is done so that the JDBC call will return the same results with an IMS database that it would return with a relational database.

Without these command codes a search on a range of root segment keys is done with logic that assumes the roots are in key sequence. A call qualified on a range of root segment keys attempts to begin the search with the key at the bottom of the range. For example, a search for roots with keys  $\geq 1000$  and  $\leq 2000$  begins by using the partition selection exit routine and/or the randomizer with key 1000. Subsequent GN or GHN calls with the same qualification move forward in the database. If a root segment with key  $> 2000$  is found, the search ends. The JDBC driver uses the new command codes to change the logic of the search. With the new command codes all root segments in the database are examined. This provides the expected results from the JDBC call. Of course, there is a performance cost for searching the entire database.

The performance cost for the search may be eliminated by using an alternative. If many of these calls might be issued by a program, one may create a secondary index on this key. Then one may use the secondary index for the calls by specifying PROCSEQ= referencing the secondary index for full function databases or PROCSEQD= referencing the secondary index for DEDBs. This will allow the search to be done without examining the root segments which are not in the key range. It avoids the sequential scan of the database. This is analogous to placing an index on a column in a relational database. In fact, with a relational database one should create the index when this type of JDBC call is made.

## ***CICS Threadsafe Support***

- **CICS 4.2 adds support for threadsafe IMS database calls with IMS 12**
  - Eliminates TCB switches for IMS database calls
    - Without threadsafe support, IMS call must be done under an IMS TCB
      - Requires switch from CICS QR TCB to IMS TCB and back to CICS QR TCB
      - If application is running under an OPEN TCB it also requires a switch from OPEN TCB to QR TCB and back from QR TCB to OPEN TCB
    - With threadsafe support, IMS call may be done under a CICS OPEN TCB
      - No TCB switch
      - CICS has multiple OPEN TCBs
        - Multiple DLI calls may be done in parallel under CICS OPEN TCBs
  - Enhancement applies to both EXEC DLI and CALL DLI
- **Benefits**
  - Lower CPU use
  - Increased throughput

651

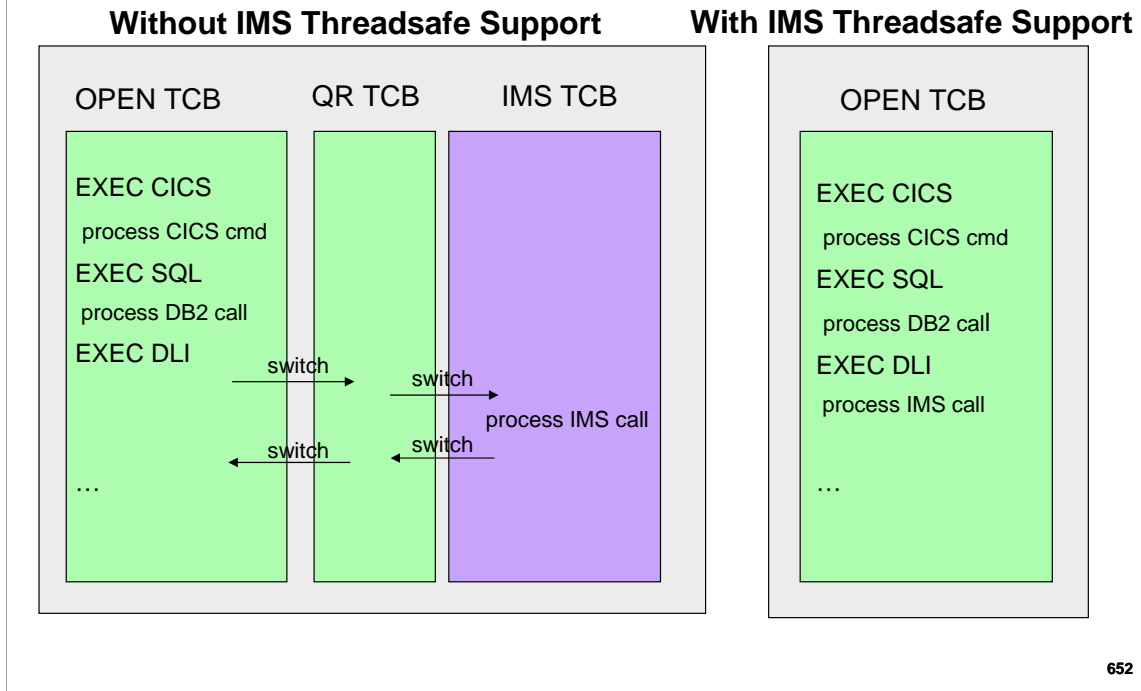
CICS Transaction Server Version 4 Release 2 includes threadsafe support for IMS database calls. This potentially eliminates TCB switches for IMS calls and saves the CPU usage associated with TCB switches.

Without threadsafe support calls for IMS databases require a switch to an IMS TCB for processing the call and a switch back to a CICS PCB when IMS completes call processing. Threadsafe support allows the IMS call processing to be done under the CICS TCB. In fact, without this support many IMS database calls from CICS require four TCB switches. This depends on the CICS application program execution environment. Without the threadsafe support for IMS, all IMS calls must be switched from the CICS QR (quasireentrant) TCB to the IMS TCB. If the application is processing under a CICS OPEN TCB, this requires a switch from the OPEN TCB to the QR TCB to the IMS TCB. When call processing is completed by IMS there are switches from the IMS TCB to the QR TCB to the OPEN TCB. This is four TCB switches for an IMS call.

Threadsafe support already exists for DB2, MQ, CICS Sockets, XPLINK and many CICS commands. This support is provided for applications executing under a CICS OPEN TCB. It provides two benefits. First, it avoids TCB switches for processing these requests. Second, it allows the concurrent dispatching of these requests. There is only one QR TCB in a CICS address space. Multiple requests running under a QR TCB cannot be dispatched concurrently. For these reasons, it may be likely that CICS transactions which use these services along with IMS calls are running under an OPEN TCB. These transactions are likely to benefit most from the threadsafe support for IMS.

Threadsafe support applies to IMS database access using either the EXEC DLI or the CALL DLI interface in CICS.

## CICS Threadsafes Support



This illustrates the difference between accessing IMS from a CICS application without the use of threadsafes support and with the use of threadsafes support.

On the left you can see a CICS application which invokes a CICS command (a CICS service), accesses DB2, accesses IMS, invokes another CICS command and accesses DB2 again. The CICS commands and DB2 accesses are done under the OPEN TCB. Without threadsafes support the IMS call first causes a switch to the QR TCB and then a switch to an IMS TCB. IMS processes the call under its TCB. After the call is processed, another switch is required to the CICS QR TCB. Finally, there is a switch to the CICS OPEN TCB.

On the right you can see the same CICS application with threadsafes support. All of the processing is done under the CICS OPEN TCB. This eliminates four TCB switches for the IMS database call.

## **IRLM 2.3**

- IRLM 2.3 and IRLM 2.2 are both shipped with IMS 12
- IRLM 2.3 and IRLM 2.2 may be used with any supported version of IMS
  - IRLM 2.3 is required by DB2 Version 10
    - IRLM 2.3 has 64-bit caller interface
      - IMS continues to use the 31-bit caller interface
  - IRLM 2.3 requires z/OS 1.10 or higher
- IRLM 2.3 provides improved performance for some requests
  - We do not expect a substantial performance improvement with IRLM 2.3 with IMS

653

Both IRLM 2.2 and IRLM 2.3 are delivered with IMS 12. Both of these IRLMs may be used with any supported version of IMS.

IRLM 2.3 is required by DB Version 10; however, IRLM 2.2 may be used by the IMS database manager when DB2 is using IRLM 2.3. IRLM 2.3 supplies a 64-bit caller interface that is required by DB2 Version 10. IMS does not use this interface.

IRLM 2.3 must run under z/OS 1.10 or higher.

IRLM 2.3 provides some improved performance; however, we do not expect substantial performance improvements with IMS.

## ***Miscellaneous Database Enhancements***

- Reuse of Local DMB Numbers
- Display status of randomizers and partition selection exit routines
- Retry after lock timeouts
- Improved information with lock timeouts
- Batch Data Sharing Abend Elimination
- RACF userid in Data Capture batch log records
- Increased VSAM pools from 16 to 255
- Temporary close of VSAM data sets when acquiring new extents
- Elimination of OSAM U0080 Open/Close/EOV Abends
- DFS993 sent to system console
- CA Reclaim Support
- New command codes for sequential search
- CICS threadsafe support
- IRLM 2.3

# HALDB Enhancements



## ***HALDB Enhancements***

- Parallel Migration to HALDB
- Optional release of HALDB OLR ownership when IMS terminates
- Reuse of HALDB partition DB names for non-HALDB databases

## ***HALDB Enhancements***

- Parallel Migration to HALDB
- Optional release of HALDB OLR ownership when IMS terminates
- Reuse of HALDB partition DB names for non-HALDB databases
- Reorganization Number Handling by Timestamp Recovery

## ***Parallel Migration to HALDB (IMS 10 and IMS 11 SPE)***

APARs: IMS 10 PM06635; IMS 11 PM06639

- **HD Unload (DFSURGU0) may be run in parallel for different key ranges**
  - Unloads for migration to HALDB
  - Applies to HIDAM, HDAM and HISAM
- **HD Reload may be run in parallel for different key ranges**
  - Reloads for migration to HALDB
- **Benefits**
  - Migration elapsed time can be reduced significantly
  - Especially important with logical relationships

658

The IMS Unload utility (DFSURGU0) has been enhanced in IMS 10, IMS 11 and IMS 12 to allow unloads of key ranges of an HDAM, HIDAM or HISAM database when migrating to HALDB. Multiple unloads for the same database may be run in parallel. This can significantly reduce the elapsed time for a migration to HALDB. This is especially important for databases with logical relationships since their unloads for migration may require a long time.

## ***Parallel Migration to HALDB***

- **The problem before this enhancement**
  - Migration of databases with many logical relationships could require a very long elapsed time
    - In most cases, when unloading a database each logical relationship requires a random read of the related database
      - These random reads account for almost all of the elapsed time of the unload
  - The unload can not be broken into multiple parallel steps
    - Unload always reads the entire database
  
- **The solution**
  - Unload can now read part of a database for a migration unload
  - Multiple unloads for different key ranges may be run in parallel
  - Multiple reloads for different partitions may be run in parallel

659

Before this enhancement the migration unload of a logically related database often took a long time. When a logical child segment is unloaded, its logical parent must be read in most cases. This is a random read. These random reads account for almost all of the elapsed time of the unload.

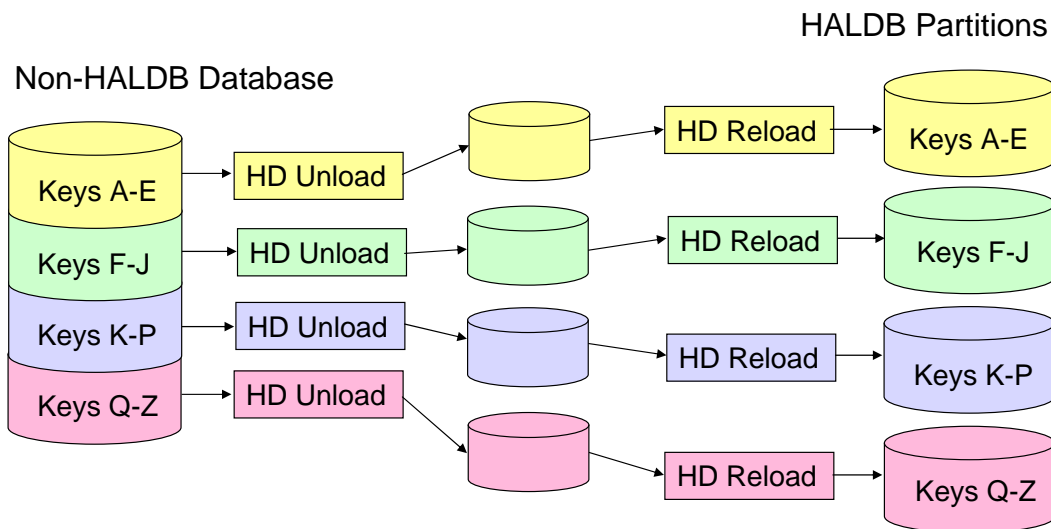
The logically related database must be read unless the unload is for the physical logical child of a bidirectional logical relationship with virtual pairing when the PHYSICAL option was specified the DBD to include the concatenated key in the physical logical child. The logical related database is always read with the following logical relationships:

- Unidirectional logical relationships
- Physically paired bidirectional relationships
- Virtually paired bidirectional logical relationships when reading the virtual logical child
- Virtually paired bidirectional logical relationships when reading the physical logical child and the VIRTUAL option has been specified in the DBD (the concatenated key of the logical parent is not stored in the physical logical child)

The solution to this problem is to allow multiple unload jobs to be run in parallel. The enhancement allows an unload job to unload part of a database. Without the enhancement, an unload always reads an entire database.

For a HIDAM database, the enhancement allows the unload to read only the database records in the key range for the root segments. For an HDAM database, all of the root segments in the database are read. Even though the unload of an HDAM database requires the reading of all of the database root segments, it is much faster than unloading the entire database when there are logical relationships. In almost all cases, the unload of a logical relationship requires the access to the logically related database. These random reads typically account for almost all of the elapsed time of the unload. When HD Unload is restricted to a key range, only the relationships from the records in the key range require the reading of the logically related database.

## Parallel Migration to HALDB



660

This picture illustrates the improved process. The four HD Unload jobs process different key ranges in the non-HALDB database. They are run in parallel. Their individual outputs are fed to four different HD Reload jobs which load the four partitions in the new HALDB database. The reload jobs are also run in parallel. This significantly reduces the elapsed time of the migration. It should be approximately one fourth the elapsed time that would be required without the parallel running jobs.

## Parallel Migration to HALDB

- Each execution of HD Unload may be limited to a key range
  - Unloads of HIDAM and HISAM read only the database records in the key range
  - Unloads of HDAM read all root segments in the database and select only the records for the specified key range
- SYSIN DD control statements:

```
MIGRATE=YES
```

```
KEYRANGE FROMKEY(key value) TOKEY(key value) KEYLEN(key lth)
```

- Recommendation: Cut and paste key values from RECON listing
- Example:

```
MIGRATE=YES
```

```
KEYRANGE FROMKEY(F0F0F0F0F0) TOKEY(D2F2F0F0F0) KEYLEN(5)
```

661

An execution of HD Unload may be restricted to a range of keys when the MIGRATE=YES control statement is included for migration to HALDB. The low (FROMKEY) and high (TOKEY) values must be specified in hexadecimal. These values may be obtained from the output of a LIST.DB DBRC command for the partition.

Multiple executions of HD Unload may be run in parallel. The key ranges should correspond to the key ranges for HALDB partitions. This allows the subsequent HD Reload executions to be run in parallel. It is highly recommended that you “cut and paste” the hexadecimal key values from a RECON listing. This potentially avoids typographical errors would might be tedious to detect.

## ***Parallel Migration to HALDB***

### ▪ **Restrictions**

- May only be used with the MIGRATE keyword.
  - Cannot be used with MIGRATX
- Unload range must match HALDB partition boundaries
  - HD Reload accepts only one input data set
- Input data set must contain all of the records for the partition(s) being loaded
  - HD Reload accepts only one input data set
  - HD Reload can load one or multiple partitions
- Concurrent unloads require DISP=SHR for database data sets
- Concurrent unloads require VSAM share options that allow multiple readers
  - SHAREOPTION(3 3) may be used

662

Note that MIGRATE=YES must be specified. The new capability cannot be used for unloading HALDB databases. Also, MIGRATX=YES may not be used with the KEYRANGE statement. MIGRATX=YES is used only for databases with secondary indexes. It cannot be used with the KEYRANGE statement since the key ranges appropriate for a secondary index would not be those used for the unload of the indexed database. If your database has secondary indexes you may use MIGRATE=YES and create the secondary indexes with a tool such as IBM IMS Index Builder.

The records unloaded must match the HALDB partition boundaries. This is true because only one output data set from HD Unload may be used as input to HD Reload. Similarly, all of the records for a partition must be unloaded with one execution of HD Unload. One can get the key values from a listing of the HALDB partitions in the RECONS.

Parallel execution of the unloads will read the same database data sets. The JCL must specify DISP=SHR for the unloads to run in parallel. If the data sets are VSAM, you must also specify share options which allow concurrent reads. SHAREOPTIONS(3 3) may be used for this.

## ***HALDB Online Reorganization (OLR) Ownership Release***

- **IMS 12 adds capability to release ownership of an OLR when IMS terminates**
  - IMS termination may be normal or abnormal
    - In previous IMS versions, OLR ownership was kept by a terminated IMS system
  - If OLR is owned by an IMS system, it may not be started or restarted on another IMS system
  
- **Benefit**
  - OLRs may be restarted on another available IMS
  - Caution:
    - If an OLR is not owned by a terminated IMS system, it will not be automatically restarted when the IMS system is restarted

663

IMS 12 provides an option for the release of ownership of a HALDB Online Reorganization when the IMS system on which it is executing terminates. The termination may be either a normal or abnormal termination of IMS. If ownership is released, the OLR may be restarted on another IMS system. If ownership is not released, the OLR cannot be restarted on another IMS system.



## HALDB OLR Ownership Release

- Specification of ownership release default

- Determined by parameter in DATABASE section of DFSDFxxx

- RELOLROWNER – specified to release ownership
- Absence of RELOLROWNER specifies that ownership is retained
  - As in previous releases

```
<SECTION=DATABASE>  
RELOLROWNER=Y | N
```

- Default may be overridden by parameter on INIT OLREORG, /INIT OLREORG, UPD OLREORG or /UPD OLREORG command

- OPTION(REL)
- OPTION(NOREL)

664

The option is specified by including a RELOLROWNER=Y statement in the DATABASE section of the DFSDFxxx PROCLIB member. RELOLROWNER=N is the default and does not release ownership when the IMS system terminates. The RELOLROWNER= value may be overridden by specifying OPTION(REL) or OPTION(NOREL) on the INIT OLREORG, /INIT OLREORG, UPD OLREORG or /UPD OLREORG command.

When RELOLROWNER=Y or OPTION(REL) is not specified, OLR is automatically restarted when the terminated IMS system is restarted. When RELOLROWNER=Y is specified OLR is not automatically restarted unless it was overridden with OPTION(NOREL) on the command. If the OLR is not automatically restarted by IMS restart, it must be restarted with the INIT OLREORG or /INIT OLREORG command.

## HALDB OLR Ownership Release

- Ownership setting is returned by QUERY and /DISPLAY commands

### - /DIS DB OLR

DATABASE	PART	RATE	BYTES	SEGS	ROOTS	STARTTIME
STATUS						
DBHDOJ01	PDHDOJA	10	53330	217	31	32110/143354
WAITRATE, OPTDEL, OPTREL						
*32110/143356*						

### - QRY OLREORG

Partition	MbrName	CC	LclStat	Rate	Bytes-Moved	Segs-Moved
POHIDKA	IMS1	0	RUNNING	100	72315678	244597
PVHDJ5A	IMS1	0	RUNNING	100	8454630	30029
Roots-Moved Option Resumed StartTime						
22511	NODEL	Y		2010.320	10:20:21.61	
3775	DEL, REL			2010.320	10:20:21.84	

665

The ownership release status is shown in the response to /DIS DB OLR an QRY OLREORG commands as shown on this slide.

## ***Reuse of HALDB partition DB names***

- **Reuse of HALDB partition DB names for non-HALDB databases**
  - IMS 12 allows names of deleted partitions to be used as non-HALDB database names
    - Previous versions of IMS did not free the DDIRs for deleted partitions
      - Required restart of IMS online system
  
- **Benefit**
  - More flexibility in the use of database names

Each database and each HALDB partition uses a DDIR control block in the IMS system. In versions previous to IMS 12 the deletion of a HALDB partition did not delete its DDIR in an online system. This prevented the reuse of the partition name as a database name. IMS 12 has changed this. The deletion of the partition will result in the deletion of its DDIR. This allows the unused partition name to become a database name

## ***Reorganization Number Handling by Timestamp Recovery***

- **IMS 12 Database Recovery utility sets reorganization number of partition based on value in RECON**
  - Previous IMS versions did not coordinate the numbers in RECON and partition data set
    - Reorg number in data set was updated from RECON value by first IMS subsystem which updated the partition
    - Index Builder tool created index pointers based on the reorg number in the data set
      - Index entries needed “healing” when reorg number was changed by updater
    - Pointer Checker reported errors
- **Benefit**
  - IMS 12 eliminates the Index Builder and Pointer Checker problems
    - Reorg number in data set matches the number in RECONS when timestamp recovery is done

667

The partition reorganization number is used to ensure that secondary index and logical relationship pointers are accurate. The reorg number for a partition is stored in the partition data set. It is incremented by each reorganization of the partition. The reorg number is also stored in the Extended Pointer Set (EPS) of secondary index entries and logical children. If the value in the EPS does not match the value in the partition data set, the pointer is healed by updating it from the Indirect List Data Set (ILDS). The reorg number is also stored in the partition database record in the RECON.

When a timestamp recovery is done to a time before the last reorganization, the reorg number in partition data set is returned to its previous value by the actions of the Database Recovery utility (DFSURDB0) in previous versions of IMS. IMS 12 changes this. The IMS 12 Database Recovery Utility takes the reorg number from the RECONS, increments it and stores it in both the RECONS and the partition data set. This makes the reorg numbers in the partition data set and the RECON match. Previously, a mismatch occurred until the first update job for the partition was executed. An update batch job or online systems takes the value from the RECON and writes it to the partition data set.

Two problems could occur with previous releases of IMS. First, if a user created indexes using the Index Builder tool after a timestamp recovery the EPSs in the index entries would contain the reorg values from the partition data set. These did not match the values in the RECON. After the first update job, the reorg number in the partition data set would be updated. This caused a mismatch with the values in the EPSs. When the index entries were used, IMS would “heal” the pointers. This was unnecessary overhead. The new process eliminates this “healing” since the values in the EPSs will match the values in the partition data set. The second problem occurred with the Pointer Checker tool. When the values in the pointers do not match those in the partition data set, the Pointer Checker produces error or warning messages. The new process eliminates these messages.

## ***HALDB Enhancements Summary***

- **Parallel Migration to HALDB**
  - New capability for HD Unload
- **Optional release of HALDB OLR ownership when IMS terminates**
  - OLRs may be restarted on another available IMS
- **Reuse of HALDB partition DB names for non-HALDB databases**
  - IMS restart is not required
- **Reorganization Number Handling by Timestamp Recovery**
  - Eliminates Index Builder and Pointer Checker problems