Application Design and Programming with HALDB

Rich Lewis
IBM
IMS Advanced Technical Support

© IBM Corporation 2002

IMS Version 7 adds High Availability Large Database (HALDB) capabilities. HALDB supports very large databases. By splitting a database into multiple partitions, HALDB allows it to contain up to 40 terabytes! In spite of this, an application program continues to see one database. That is, the database is addressed by one PCB.

In general, application programs do not have to be modified when a database is migrated to HALDB. On the other hand, there are cases where application programs must be changed. Also, an installation may want to take advantage of some capabilities which may require application changes.

This presentation describes changes that you may need to make or want to make when a database is migrated to HALDB. Considerations for processing partitions in parallel, processing secondary indexes as databases, initially loading HALDB databases, handling unavailable partitions, and converting from user partitioning are explained.

IMS Version 7

Topics

- **Introduction**
  - Highlights of HALDB
  - Partitioning
- **Application Considerations**
  - Initial Loads
  - Processing Partitions in Parallel
  - Restricting a PCB to One Partition
  - Handling Unavailable Partitions
  - Processing Secondary Indexes as Databases
  - Converting from User Partitioning

© IBM Corporation, 2002

IMS Version 7

Highlights of HALDB

© IBM Corporation, 2002

This section briefly describes the characteristics and benefits of HALDB.

IMS Version 7

HALDB (High Availability Large Database)

Large Database

Up to 10,010 data sets per database!

Greater than 40 terabytes

- Databases are partitioned
  - Up to 1001 partitions per database
  - Partitions have up to 10 data set groups

High Availability Database

- Partition independence
  - Allocation, authorization, reorganization, and recovery are by partition
- Simplified and shortened reorganization process
  - Partitions may be reorganized in parallel
  - Reorganization of partition does not require changes to secondary indexes or logically related databases which point to it
  - Prefix Resolution, Prefix Update, and secondary index rebuilds are eliminated

© IBM Corporation, 2002

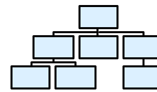
IMS V7 introduces a new capability for full function databases. This is High Availability Large Database (HALDB). HALDB databases have up to 1001 partitions. Each partition has up to 10 data set groups. This gives HALDB up to 10,010 data sets per database. Each of these data sets may be up to 4 gigabytes. So, the limit is 40 terabytes per database.

HALDB provides two availability benefits. First, partitions are managed independently. Each partition in a database may be allocated, authorized, reorganized, and recovered independently. Second, the reorganization of the database is much faster. First, multiple partitions allow users to reorganize and image copy smaller amounts of data. This takes less time. The reorganizations may be done in parallel as can the image copies. Second, the reorganization of a partition does not require utilities to update the pointers in secondary indexes and logically related databases which point to the reorganized data. Even though the reorganization moves segments, pointers to those segments are not updated by the reorganization process. Instead, these pointers are updated as needed. This is a "self healing" process. This combination of capabilities can greatly reduce the windows required for database maintenance. There is no need for utilities such as Prefix Resolution and Prefix Update to update pointers. There is no need to rebuild secondary indexes. This also reduces the time required for reorganizations.

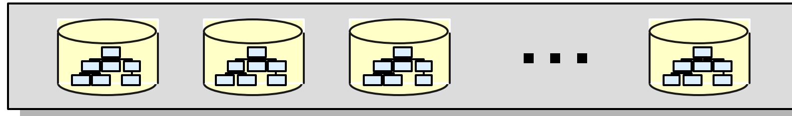
IMS Version 7

Highlights

- **New database types**
  - PHDAM - partitioned HDAM
  - PHIDAM - partitioned HIDAM
    - Index is also partitioned
  - PSINDEX - partitioned secondary index
- **Hierarchic structure is maintained**
  - A database record resides in one partition



- **Partition selection (deciding in which partition a record resides)**
  - By key range or by user exit routine



© IBM Corporation, 2002

HALDB introduces three new full function database types, partitioned HDAM (PHDAM), partitioned HIDAM (PHIDAM), and partitioned secondary index (PSINDEX). As the names imply, these are partitioned versions of the corresponding database types for non-HALDB databases. PHIDAM includes its index which is also partitioned.

HALDB databases have the same hierarchic structure that is used for other full function databases. A HALDB database record, which is a root segment and all of its dependents, resides in one partition.

Partitioning may be done either by key range or by a user written exit routine. Either method may be used with each of the three database types, PHDAM, PHIDAM, and PSINDEX.

IMS Version 7

Highlights

- **Logical relationships and secondary indexes are supported**
  - Secondary indexes may be partitioned
- **Parallel Processing**
  - Reorganizations
    - Partitions may be reorganized independently
    - Partitions may be reorganized in parallel
  - Application processing
    - Partitions may be processed in parallel
    - DBRC authorization is by partition (not entire database)

© IBM Corporation, 2002

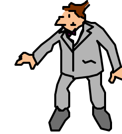
HALDB has full support of logical relationships and secondary indexes. The secondary indexes may be partitioned.

HALDB provides for the possibility of parallel processing. Partitions of databases may be reorganized with separate jobs. These jobs may be run in parallel. The capability to reorganize partitions of a database in parallel with multiple jobs can greatly reduce the time required to reorganize a database. Application processing against different partitions may be done in parallel. DBRC authorization is by partition, not an entire database. This means that different batch jobs may process different partitions of the same database without requiring data sharing.

IMS Version 7

The Application Programming News

- **The good news:**
  - In general, application programs do not have to be changed when databases are migrated to HALDB



- **The OK news:**
  - You may want to change application programs to take advantage of new opportunities with HALDB

- **The not so good news:**
  - You may have to change a small number of application programs when databases are migrated to HALDB

© IBM Corporation, 2002

For application programming there is a lot of good news. In general, existing application programs do not have to be changed when a database is migrated to HALDB.

On the other hand, you may want to make some application changes. Since HALDB partitions may be managed and processed independently, you may want to modify applications to take advantage of this capability.

There are a few cases where existing application programs must be modified when a database is migrated to HALDB. We will see that these cases are rare. They occur in two situations. First, some program which initially load databases with logical relationships may have to be changed. Second, some programs which process secondary indexes as if they were databases, may have to be changed. We will see the exact circumstances in which changes are required.

Partitioning

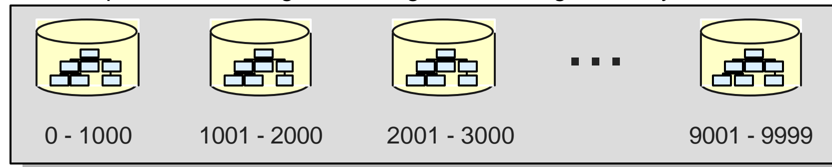
© IBM Corporation, 2002

This section describes the ways in which HALDB databases may be partitioned and how partitioning affects the order of segments in a database.

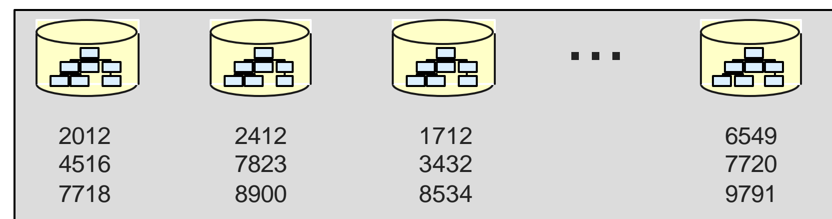
IMS Version 7

Partitioning Choices

- **Two methods of partitioning**
  - Key range
    - Each partition is assigned a range of root segment keys



  - Partition Selection Exit routine
    - The exit routine assigns a root segment to a partition based on its key



© IBM Corporation, 2002

HALDB databases may be partitioned using either of two methods. In both methods the key of the root segment is used to assign a database record (a root segment and all of its dependents) to a partition. Either method may be used with each of the HALDB database types, PHDAM, PHIDAM, and PSINDEX.

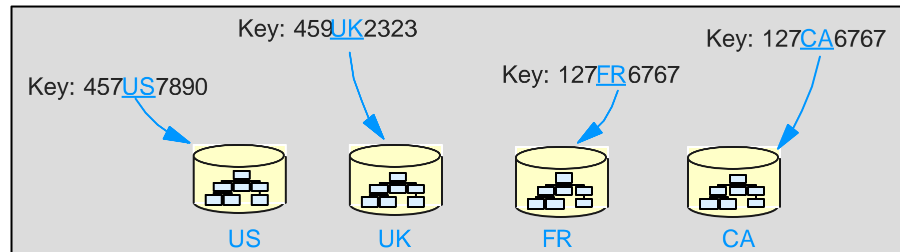
Key range partitioning assigns a range of keys to a partition. For each partition we specify the highest key that may reside there. In this example, the first partition has a high key of 1000. The second partition has a high key of 2000, etc. So, all keys from 0 to 1000 will be placed in the first partition. Keys from 1001 to 2000 will be placed in the second partition.

A Partition Selection Exit routine may be used to assign keys to partitions. With this method, an exit routine is specified for the database. This exit routine makes the partition selection decision for each key. It can use whatever method its programmer chooses to use. In this example, keys 2012, 4516, and 7718 are assigned to the first partition. Keys 2412, 7823, and 8900 are assigned to the second partition. It is not clear how the exit routine has made these decisions.

IMS Version 7

Partitioning Choices

- **Partition Selection Exit routine example:**
  - Assign records by country code which is in the root key



© IBM Corporation, 2002

One might use a Partition Selection Exit routine if a low order part of the key has information which is useful in assigning records to partitions. In this example, the fourth and fifth bytes of the key include a country code. The installation wants to create a separate partition for each country. The exit routine examines the fourth and fifth bytes of the key and assigns records based on these bytes.

IMS Version 7

Order of Segments in a Partition

- **PHDAM - Partitioned HDAM**
  - Roots are in random order within a partition
- **PHIDAM - Partitioned HIDAM**
  - Roots are in sequential order within a partition
- **PSINDEX - Partitioned Secondary Index**
  - Entries are in sequential order within a partition

© IBM Corporation, 2002

The order of records (root segments) in a partition depends on the database type.

As we would expect, PHDAM records are stored randomly. This depends on the database randomization routine. Each partition may have its own randomization routine. Typically, we could use DFSHDC40 just as we would use it for a non-HALDB HDAM database. Sequential processing of a partition will return root segments in a random order. That is, they will not be in key sequence.

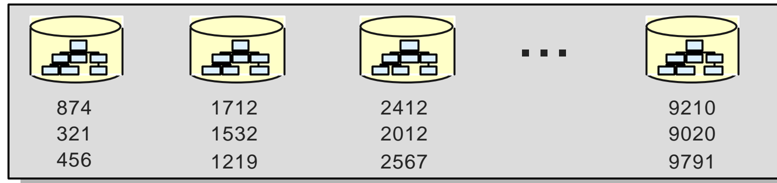
As we would expect, PHIDAM records are stored sequentially within a partition. Sequential processing within a partition will return PHIDAM root segments in key sequence.

PSINDEX entries are also stored in key sequence. If we process a database using a secondary index processing sequence, the records will be returned in secondary index key sequence within a secondary index partition.

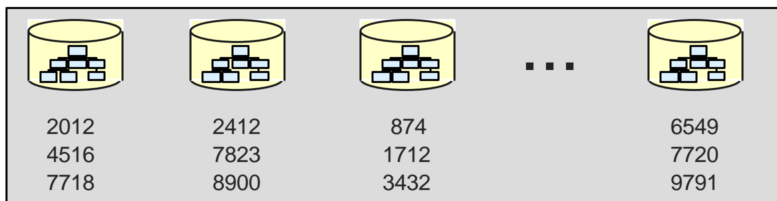
IMS Version 7

Order of Segments in a PHDAM Database

- **PHDAM with key range partitioning**
  - Keys are sequential between partitions, random within a partition



- **PHDAM with Partition Selection Exit routine**
  - Keys are not sequential between partitions, random within a partition



© IBM Corporation, 2002

Let's look at the order of segments in a database, not just within a partition.

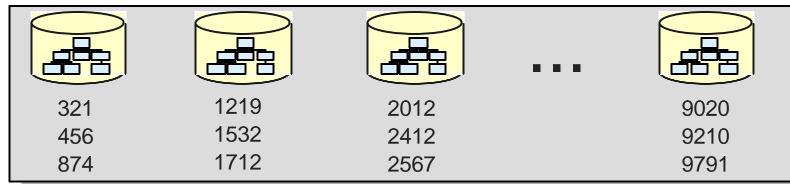
If we use key range partitioning with PHDAM, keys are sequential between partitions. This means that the lowest range of keys will be in the first partition. The next range of keys will be in the second partition, etc. Of course, within any partition the keys will be in a random order determined by the randomization routine. In this example, keys from 0-1000 are randomly ordered in the first partition. Keys from 1001 to 2000 are randomly ordered in the second partition, etc.

If we use a Partition Selection Exit routing with PHDAM, the keys are not sequential between partitions. The exit routines determines which are placed in which partition. Of course, within any partition the keys are in random order. The randomization routine determines the order.

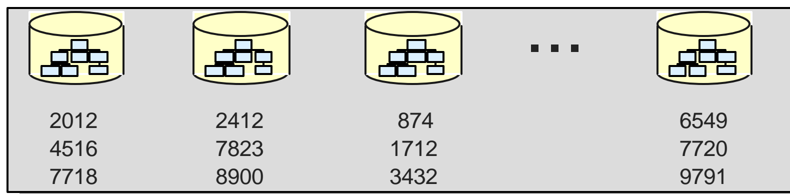
IMS Version 7

Order of Segments in a PHIDAM Database

- **PHIDAM with key range partitioning**
  - Keys are sequential between partitions, sequential within a partition



- **PHIDAM with Partition Selection Exit routine**
  - Keys are not sequential between partitions, sequential within a partition



© IBM Corporation, 2002

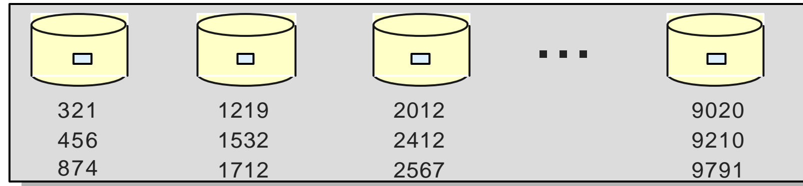
If we use key range partitioning with PHIDAM, keys are sequential between partitions and across partitions. The lowest range of keys will be in the first partition and they are sequential within the partition. Similarly, the next range of keys will be in the second partition and they are sequential within the partition. This means that sequential processing of the database will return the root keys in key sequence. This is the same as the sequential processing of a non-HALDB HIDAM database.

If we use a Partition Selection Exit routing with PHIDAM, the keys are not sequential between partitions. The exit routines determines which are placed in which partition. Of course, within any partition the keys are in sequential order.

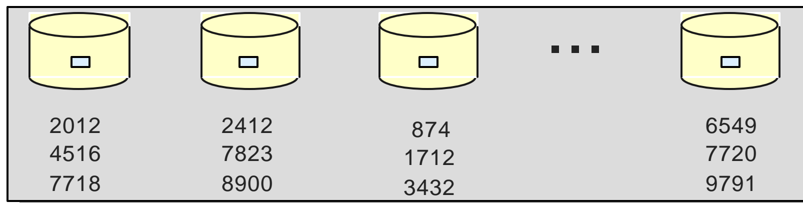
IMS Version 7

Order of Segments in a PSINDEX Database

- **PSINDEX with key range partitioning**
  - Keys are sequential between partitions, sequential within a partition



- **PSINDEX with Partition Selection Exit routine**
  - Keys are not sequential between partitions, sequential within a partition



© IBM Corporation, 2002

PSINDEX is much like PHIDAM.

If we use key range partitioning with PSINDEX, keys are sequential between partitions and across partitions. The lowest range of keys will be in the first partition and they are sequential within the partition. Similarly, the next range of keys will be in the second partition and they are sequential within the partition. This means that using the secondary index as a processing sequence will result in the retrieval of the records in the indexed database in the secondary index key sequence. This is like using a non-HALDB secondary index as a processing sequence.

If we use a Partition Selection Exit routine with PSINDEX, the keys are not sequential between partitions. The exit routine determines which are placed in each partition. Of course, within any partition the keys are in sequential order.

IMS Version 7

Application Considerations

- Initial loads
- Processing Partitions in Parallel
- Restricting a PCB to One Partition
- Handling Unavailable Partitions
- Processing Secondary Indexes as Databases
- Converting from User Partitioning

© IBM Corporation, 2002

Now that we have the necessary background information, we can look at the considerations for application design and coding when we migrate a database to HALDB.

We will look at initial loads of HALDB databases. This includes the required order of records when loading a database, how to load partitions in parallel, and how to handle logical children. Logical children in HALDB databases cannot be created by initial load.

We will look at processing partitions in parallel. If we process in parallel, we may need to limit the processing of a program to a single partition. We will see how to do this.

We will look at how we may handle unavailable partitions and what we must do to process secondary indexes as databases.

Finally, we will consider the conversion of user partitioned databases. User partitioning is the term to describe the use of multiple databases as if they were partitions of a database. Many installations have used this technique in the past. We will see what they have to do when such a set of databases is migrated to a single HALDB database.

IMS Version 7

Initial Loads

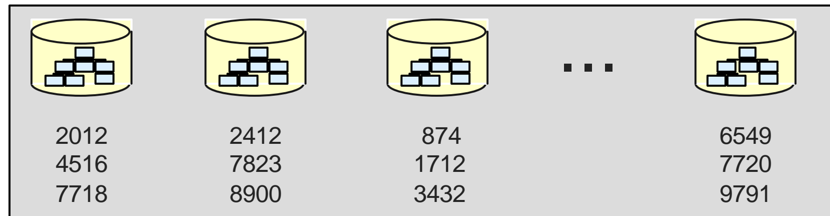
© IBM Corporation, 2002

This section describes the initial loads of HALDB databases. It does not discuss the migration of a non-HALDB database to HALDB. Migrations are not covered in this presentation.

IMS Version 7

Initial Loads

- **Initial load uses PROCOPT=L or PROCOPT=LS in PCB**
  - Same as non-HALDB databases
- **Loads of PHDAM roots may be in any key sequence**



- Could load 874, 1712, 2012, 2412, 4516, 6549, 7718, 7720, ...
- Could load 8900, 2012, 7823, 4516, 7718, 2412, 874, 1712, ...
- No changes required for load program
  - If you sort by RAP sequence for faster loads, you will want to sort by RAP sequence within partitions

© IBM Corporation, 2002

Initial loads of HALDB databases use either PROCOPT=L or PROCOPT=LS in the database PCB. This is the same as is used for non-HALDB databases.

For PHDAM databases, initial load may use PROCOPT=L. There is no requirement for any order of the roots segments. They may be loaded in any sequence.

This example shows that the roots could be loaded in key sequence. They could be loaded in an arbitrary sequence, such as 8900, 2012, 7823, etc. Any program that successfully loads an HDAM database may also be used to successfully load the database after it has been converted to HALDB. There is one exception to this rule. The exception deals with logical relationships, not the order of the segments. It will be discussed later in this presentation.

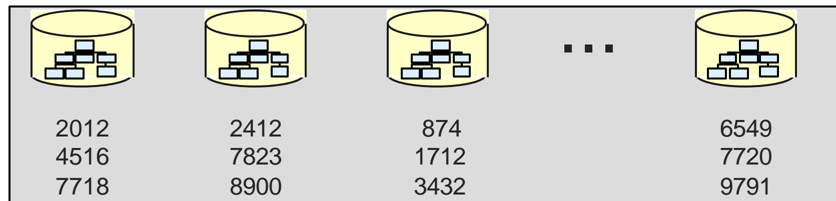
Many installations load HDAM databases in RAP sequence. This loads the records in physical sequence in the database. This is done by sorting the input records into the order that the randomization routine creates. To do the same thing with PHDAM, one would have to sort the records within each partition. With PHDAM, randomization is done within a partition, not across partitions. That is, the partition is selected first and then randomization is done.

The IBM High Performance Load tool includes a Physical Sequential Sort for Reload (PSSR) program which will do this sort. That is, it will sort the records for partitions in RAP sequence. It includes HALDB support to sort records for one partition, a set of partitions, or all partitions in a database.

IMS Version 7

Initial Loads

- **Loads of PHIDAM roots <u>must</u> be in key sequence <u>within a partition</u>**
  - <u>May</u> be in key sequence <u>across the entire database</u>



- Could load 874, 1712, 2012, 2412, 4516, 6549, 7718, 7720, ...
  - This is like HIDAM database order
  - <u>No changes required</u> for load program
- Could load 2012, 4516, 7718, 2412, 7823, 8900, 874, 1712, ...

© IBM Corporation, 2002

For PHIDAM databases, initial load must use PROCOPT=LS. The roots must be loaded in key sequence within each partition. They do not have to be in key sequence across the entire database. Non-HALDB HIDAM requires that roots be loaded in key sequence across the entire database. This means that the program used to load a HIDAM database may be used to load a PHIDAM database. On the other hand, a different order of root segments could be used.

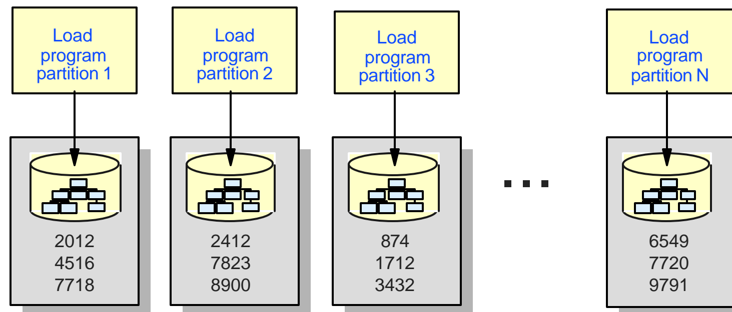
This example shows that roots could be loaded in key sequence for the entire database: 874, 1712, 2012, etc. This would be the order in which a HIDAM database would be loaded.

They also could be loaded in partition sequence: 2012, 4516, 7718, 2412, 7823, etc.

IMS Version 7

Initial Loads in Parallel

- Partitions may be initially loaded in parallel
  - Separate jobs for each partition
    - Multiple jobs cannot load records in the same partition
  - Could make load of database much faster
  - Current load programs may work without change
    - Input to program would have to be split by partition



© IBM Corporation, 2002

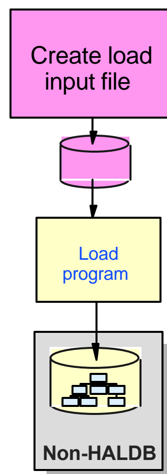
To shorten the time to load a HALDB database, you may want to load the partitions in parallel. You may use separate concurrently executing jobs to do this. Typically, each job would load a partition. This is not required. The restriction is that two or more jobs cannot load the same partition.

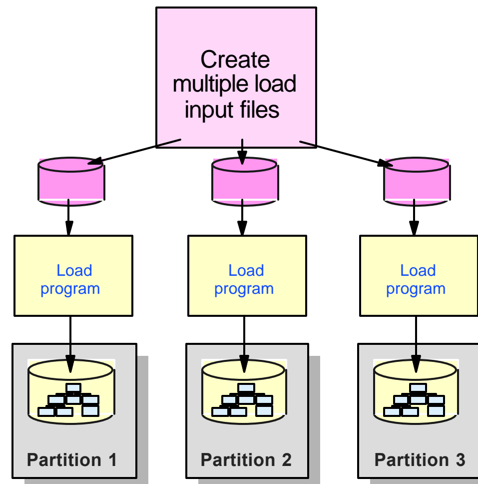
If there are many jobs, the elapsed time for loading a database could be significantly shortened. The database load would take only as long as the longest job. If it takes an hour to load of a 20 partition database with one job, loading it with 20 jobs might reasonably take less than 5 minutes.

Programs that currently load an entire database may work without change to load partitions in parallel. This is reasonable if the input to the program can be split into multiple inputs with one input for each load job.

Initial Loads in Parallel

- **Non-HALDB process**
  - Step 1 creates file read by load program
- **HALDB parallel load process**
  - Step 1 creates a file for each partition
  - Load program is unchanged





© IBM Corporation, 2002

This page shows a typical conversion to parallel loads.

For the non-HALDB database, there is an input file that drives the load program. It contains the data to be loaded in the database.

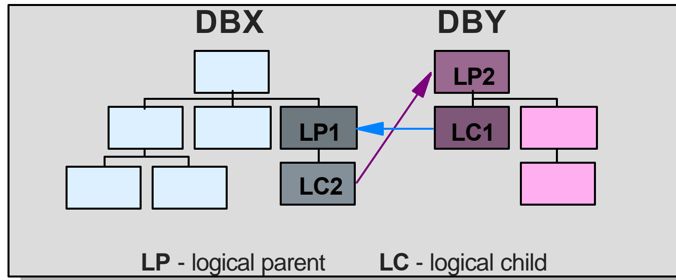
To convert this process for parallel loads, we only need to split the input file input multiple input files. We need one file for each partition. For key range partitioning this is very easy. We just create a file for each range of root keys. If we use a Partition Selection Exit routine, we have to use the same logic that the exit routine uses when we assign a record to a file.

The load program does not have to be changed. We execute multiple instances of it concurrently. The load programs do not have to "know" which partitions they are loading. They just do the same inserts that they would do for a non-HALDB database. Since the first programs only reads records for the first partition, it will only write to it. Similarly, the second program will only read records for the second partition and only write to it, etc.

IMS Version 7

Initial Loads with Logical Relationships

- **Logical relationships**



- **Logical children cannot be loaded in HALDB**
  - Attempts to load logical children receive 'LF' status code
  - They must be added by update programs (PROCOPT=I or A)
    - Cannot insert logical child without logical parent

© IBM Corporation, 2002

HALDB supports logical relationships, but it has a restriction when loading databases with logical relationships. An initial load program cannot insert a logical child segment. That is, logical children cannot be inserted with a PROCOPT of L or LS. Instead, they must be added by an update program using a PROCOPT of I or A. Attempts to insert a logical child during initial load will receive an 'LF' status code.

You cannot insert a logical child without its logical parent. If you are using an insert rule of P for the logical parent, the logical parent must exist before you insert the logical child. If you are using an insert rule or L or V, you may insert the logical parent with the logical child.

This illustration shows a bidirectional logical relationship. HALDB supports two types of logical relationships: (1) unidirectional logical relationships and (2) bidirectional logical relationships with physical pairing. HALDB does not support bidirectional logical relationships with virtual pairing.

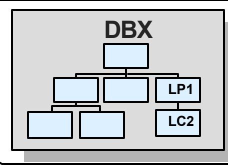
In the illustration, LC1 is a logical child pointing to its logical parent LP1. Similarly, LC2 is a logical child pointing to its logical parent LP2. LC1 and LC2 are paired logical children.

IMS Version 7

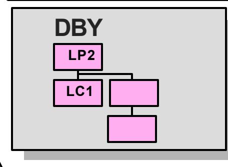
Initial Loads with Logical Relationships

Non-HALDB process:

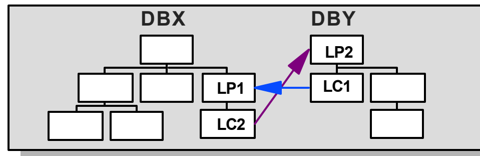
- Step 1: Prereorganization utility
- Step 2: Initial load DBX including logical children
  - Creates work file for logical relationships



- Step 3: Initial load DBY including logical children
  - Creates work file for logical relationships
- Step 4: Prefix Resolution utility
  - Sorts work file records



- Step 5: Prefix Update utility
  - Updates prefixes (pointers and counters) used for logical relationships



© IBM Corporation, 2002

This page shows the load process for non-HALDB databases with logical relationships.

Step 1 is the Prereorganization utility. It creates a control data set which is used to control the execution of later steps.

Step 2 is the initial load of database DBX. In includes the insertion of the logical parent segments (LP1) and logical child segments (LC2). The load creates a work file.

Step 3 is the initial load of database DBY. In includes the insertion of the logical parent segments (LP2) and logical child segments (LC1). The load creates a work file.

Step 4 is the Prefix Resolution utility. The inputs to the utility are the work files from steps 2 and 3. It also reads the control data set created in step 1. Prefix Resolution sorts records and resolves logical relationships. It prepares an output file for step 5.

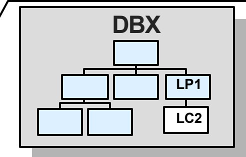
Step 5 is the Prefix Update utility. It puts pointer information in the logical children, LC1 and LC2, so that they point to their logical parents. It also puts the correct counter information in the logical parents. The counters contain a count of the number of pointers pointing to the logical parent.

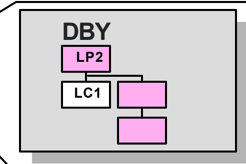
IMS Version 7

Initial Loads with Logical Relationships

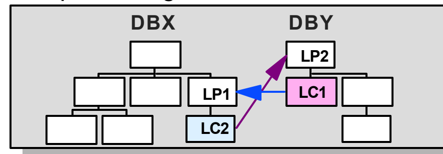
Initial load programs for HALDB with logical children must be split

- Step 1: Initialize DBX partitions with Prereorg.
- Step 2: Initialize DBY partitions with Prereorg.
- Step3 : Initial load DBX without logical children
  - This is a modification to the existing load program
- Step 4 : Initial load DBY without logical children
  - This is a modification to the existing load program





- Step 5: Insert (PROCOPT=I or A) logical children in DBX or DBY
  - New program to insert only the logical children in one database
  - Insert will create paired logical child in other database



© IBM Corporation, 2002

This page shows the load process for HALDB databases with logical relationships. There are no work files are used. The first two steps initialize the partitions in the two databases. This is done by a new function of the Prereorganization utility.

Step 3 is the initial load of database DBX. In includes the insertion of the logical parent segments (LP1) but the logical child segments (LC2) are not inserted. This could be a modified version of the load program used with the non-HALDB DBX database. The only modification required is the elimination of the insertions of the logical children. As we will see in step 5, you may want to make other modifications to this program or the one used in step 4.

Step 4 is the initial load of database DBX. In includes the insertion of the logical parent segments (LP2) but the logical child segments (LC1) are not inserted. This could be a modified version of the load program used with the non-HALDB DBY database. The only modification required is the elimination of the insertions of the logical children.

Step 5 creates the logical relationships. It is an update job which inserts the logical children. This would be a new program. Since the children are paired, they do not have to be inserted twice. The update program inserts one of the pair and IMS creates the paired logical child. The insertion of the logical children creates the pointers in the logical children and updates the counters in the logical parents. This program could get its input from either the program used in step 3 or the program used in step 4. It could use the information that was used to load the logical children in the non-HALDB version of either database. The information would include both the logical child segments and their position in the database. This information could be written to a file which could be used by the program in this step.

Processing Partitions in Parallel

© IBM Corporation, 2002

This section describes how different partitions may be processed by application programs which execute concurrently.

IMS Version 7

Online, Batch, and Data Sharing

- **Online**
  - All regions or threads have concurrent access to all partitions
  - Parallel processing is easily done (e.g. BMPs)
- **Data sharing**
  - All sharing subsystems have concurrent access to all partitions
    - Subsystems may be online subsystems or batch jobs
  - Parallel processing is easily done (e.g. DLI batch jobs)
- **Batch (without data sharing)**
  - Multiple batch jobs may have read authorization for the same partitions
  - Only one batch job can have update authorization for a partition
  - Different batch jobs may concurrently update different partitions

© IBM Corporation, 2002

In an online system, all regions and threads in the online system have access to all of the partitions that are available to the system. This makes parallel processing easy to do. For example, different BMPs could process different partitions. If a BMP occasionally needed to access data in another partition, it is free to do so.

Data sharing provides flexibility similar to that for online systems. Each subsystem can have concurrent access to all partitions. For example, different data sharing DLI batch jobs could process different partitions. If a batch job occasionally needed to access data in another partition, it is free to do so.

Batch jobs (DLI or DBB) which are executed without data sharing are more restricted. DBRC authorizations can limit what a batch job may do. Multiple batch jobs may have concurrent read authorization to the same partitions. But, only one batch job may have update authorization for a partition. If it has this update authorization, no other batch job or online system may have concurrent read or update authorization. Different batch jobs may concurrently update different partitions. But, they may not access partitions which are authorized for update to another batch job or online system. We will see that this may limit some parallel processing possibilities.

IMS Version 7

Two Styles of Batch Programs

Random and Skip Sequential

- Records are accessed by key
- Get calls are qualified on keys
  - Keys are known before calls are made
  - Program accesses only some of the records in the database
  - Program may be able to determine the partition before the call is made

Sequential

- Records are accessed by Get Next calls
- Get calls are not qualified on keys
  - Keys are not known before calls are made
  - Program typically accesses all of the records in the database
  - Program cannot know if the next record is in the same partition

© IBM Corporation, 2002

There are two styles of batch programs that could be used for parallel processing.

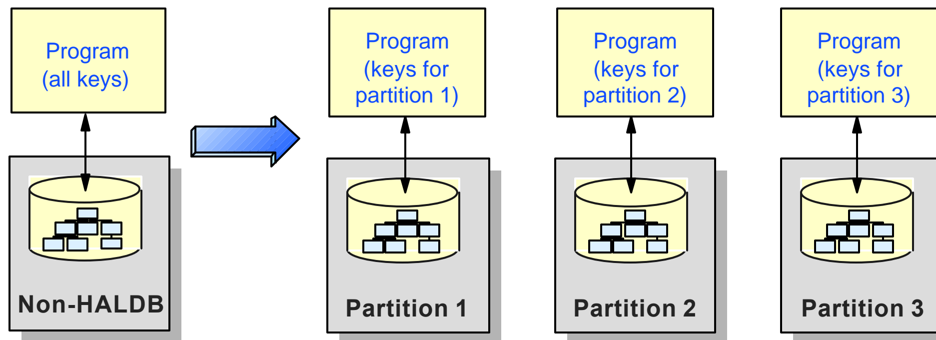
The first style uses either random or skip sequential processing. These programs access database records by the keys of the root segments. Their calls are qualified on these keys. These programs do not access all of the records in a partition. They access only the records which they request. Before they do a call, it is possible to determine in which partition the requested record resides.

The second style uses sequential processing. The records are accessed by Get Next calls without qualification on the root key. These programs attempt to access all of the records in a partition. In some cases, they may not access all records, but they access all records within a key range. The important characteristic of these programs is that they do not know which record they will get as the result of a request. They just request the next record in a database. The program does not know if the requested record is in the same partition as the last record or in a following partition.

IMS Version 7

Random or Skip Sequential Within a Partition

- Program could be modified to restrict accesses to one partition
  - Restrict to the set of keys held by the partition



© IBM Corporation, 2002

The first program style that was mentioned on the last page was random or skip sequential. It is relatively easy to confine such a program to a partition. It only needs to confine its requests to the records that reside in a partition. If the requests are coming from an external source, such as a file, we can split the source into multiple sources, one for each partition. If the requests are generated by the program, it would need to be modified to generate requests for only one partition. In either case, we can run multiple of these programs concurrently, each one accessing one partition.

These programs could be run as batch jobs without data sharing. Of course, they could be run with data sharing and/or as BMPs.

IMS Version 7

Sequential Within a Partition

- **Current program sequentially accesses entire database**
  - Accesses all records by doing GN calls
    - Begins with unqualified GN
    - Ends when 'GB' status code is returned (end of database)
- **Either**
  - Program must be modified to restrict accesses to one partition
    - Must start with first record in partition
    - Must recognize the end of the partition
- **Or**
  - HALDB Control Statement may be used
    - Restricts a PCB to a single partition

© IBM Corporation, 2002

The second style of processing is sequential. The program typically begins with an unqualified GN call to retrieve the first record in the database. It continues with these GN calls until it receives a 'GB' status code indicating the end of the database.

There are two ways to restrict a program of this style to a partition.

First, it may be modified. It must know how to find the first record in the partition. It must be able to recognize when it has reached the end of its partition. Typically, this is difficult to do. There are several potential problems. For example, the program is likely to recognize that it has read the last record in a partition when it attempts to read one from the following partition. It requires DBRC authorization to read the next partition. This could create authorization failures when batch jobs without data sharing are used. Another example is the use of PHDAM. A special randomizer would have to be used to ensure that processing began at the first RAP in a partition. For these types of reasons, a modification of existing an program may be difficult or infeasible.

Second, it may use a HALDB control statement. This statement restricts a PCB to a single partition within a database. This technique solves the problems that occur with the first technique.

IMS Version 7

Restricting a PCB to One Partition

© IBM Corporation, 2002

This section describes how the HALDB control statement may be used to limit processing to a single partition.

IMS Version 7

The HALDB Control Statement

- **Enhancement to IMS V7 and V8**
  - APAR PQ57313 for IMS V7
  - APAR PQ58600 for IMS V8
- **Control statement to limit PCB access to one partition**
  - Batch (DLI or DBB), BMP, or JBP region
  - Supported with PHDAM, PHIDAM, and PSINDEX
- **DFSHALDB DD statement:**

```
HALDB PCB=(nnn,pppppppp)

   nnn - DBPCB number
   pppppppp - partition name
```

© IBM Corporation, 2002

The HALDB control statement was added to IMS V7 by PQ57313 and to IMS V8 by PQ58600.

The control statement may be used in any type of batch job. This includes "stand alone" batch (DLI or DBB) jobs, BMP regions, and JBP regions. JBP regions are Java Batch Program regions. They were added to IMS in IMS V7.

The control statement is used to restrict a PCB to one HALDB partition. It may be used with any type of HALDB database. This includes PHDAM, PHIDAM, and PSINDEX.

The control statement identifies the PCB by its relative order in the PSB's database PCBs. It identifies the partition by its partition name.

IMS Version 7

Using the HALDB Control Statement

- **Request for first segment in database, returns first segment in partition**
  - Example:
    - Unqualified GN call with no previous position
- **GN request which reaches end of the partition, returns 'GB' status code**
  - Example:
    - Unqualified GN call with position on the last database record in the partition
- **Request for segment in another partition returns 'FM' status code**
  - Example:
    - Restriction is to partition with keys 1,000,000 to 1,999,999
    - GU call qualified with 'Root Key = 2,500,000'

© IBM Corporation, 2002

When the control statement is used, a request for the first segment in the database returns the first segment in the specified partition. For example, a program which begins with an unqualified GN call will receive the first segment in the partition.

A call which causes IMS to reach the end of the partition will return a 'GB' status code. This is the status code that normally indicates that the end of the database has been reached.

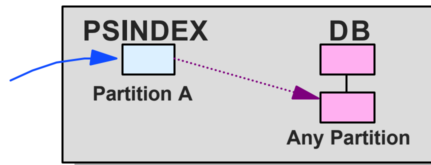
A call which would cause IMS to look in another partition for the segment returns an 'FM' status code. Examples are get or insert calls requesting a record which resides in another partition. When the HALDB control statement is not used, there are other reasons for this status code. They include a return code from a partition selection exit routine which indicates that there is no partition for the root key. Another reason for 'FM' is an attempt to insert or access a record whose root key is greater than the highest key defined for any partition when key range partitioning is used. So, the general meaning of the 'FM' status code is that the call attempted to read or insert a database record which has an invalid key.

IMS Version 7

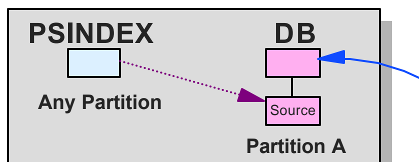
Using the HALDB Control Statement

Secondary Index usage

- When PCB specifies PROCSEQ,
  - PSINDEX partition is specified in the control statement
  - Limitation is to the PSINDEX partition, not to an indexed database partition



- When PCB does not specify PROCSEQ,
  - Database (not sec. index) partition is specified in the control statement
  - Updates to source segments may cause updates to any PSINDEX partition



© IBM Corporation, 2002

A HALDB control statement may be used with secondary indexes. It restricts the PCB to either one partition in the secondary index or one partition in the indexed database.

If the PCB includes a PROCSEQ= parameter, the control statement restricts the PCB to one partition of the secondary index. This partition may have pointers to multiple partitions in the indexed database. The control statement does not limit which partitions in the indexed database are accessed.

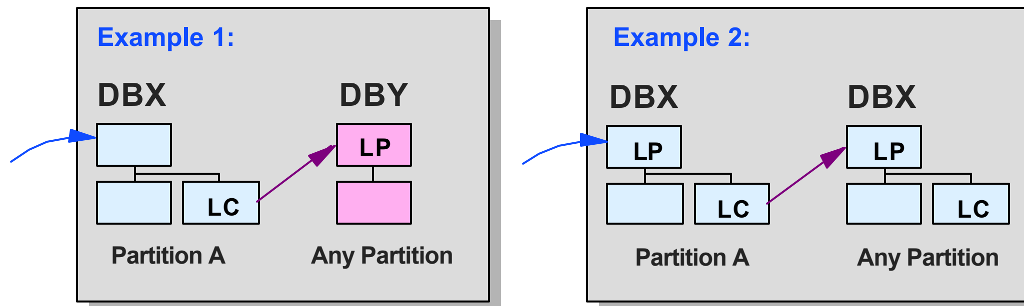
If the PSB does not include the PROCSEQ= parameter, the control statement restricts the PCB to one database of the indexed database. Updates to source segments in the indexed database may cause updates to any of the partitions in the secondary index or indexes.

For these reasons, the presence of secondary indexes will often make it impossible to process partitions in parallel when data sharing is not used. This is a characteristic of secondary indexes, not the implementation of the HALDB control statement. This is a result of the fact that the partitioning of an indexed database and the partitioning of its secondary indexes will usually be on different boundaries.

IMS Version 7

Using the HALDB Control Statement

- **Logical relationship usage**
  - Restricts access to partition in which LC resides
  - Does not restrict access to partitions in which LPs reside



© IBM Corporation, 2002

A HALDB control statement may be used with logical relationships. When this is done, it restricts the PCB to one partition in the database that is first entered by calls. This is the partition which contains the logical child segment. It does not restrict the PCB to a partition in the database containing the logical parent segments.

In the first example, the call enters database DBX. DBX contains the logical child segments. The logical parents reside in database DBY. Calls using the PCB may access any partition in DBY,

In the second example, the logical relationship is recursive. This means that the logical child and logical parent segments reside in the same database. The control statement will limit the processing to the logical children in Partition A, but logical parents in any partition may be accessed. In this case, the PCB may access any partition in the database.

For these reasons, the presence of logical relationships will often make it impossible to process partitions in parallel when data sharing is not used. This is a characteristic of logical relationships, not the implementation of the HALDB control statement. This is a result of the fact that the partitioning of the logically related databases will usually be on different boundaries or the relationships will be between segments in different partitions of the same database.

Using the HALDB Control Statement

- **HALDB control statement may be used with any batch job**
  - DBB, DLI, BMP, or JBP
- **HALDB control statements may be used for multiple PCBs**
  - One statement per HALDB PCB
- **Sequential processing**
  - May not require any application program changes
  - Could require processing to consolidate information, such as reports, from multiple executions
- **Random or skip sequential**
  - Probably requires application program changes to handle 'FM' status code

© IBM Corporation, 2002

HALDB control statements may be used with any type of batch job.

There may be control statements for multiple PCBs, but there can only be one control statement per PCB. This means that a PCB is restricted to one partition, but different PCBs may be restricted to different partitions.

Some applications which do sequential processing may not require any changes to take advantage of the HALDB control statement. The control statement will simply limit their processing to a single partition. On the other hand, some changes may be needed or desired. For example, if the batch program creates a report, the report would now be generated from the data for only one partition. Multiple executions for different partitions would produce multiple different reports. You might need to consolidate these reports. Similarly, batch programs which produce output files for subsequent processing would probably need to have these files consolidated.

Most applications which do random or skip sequential processing will probably need to be modified to take advantage of the HALDB control statement. Existing programs probably access multiple partitions. When the control statement is used, attempts to access other partitions will result in 'FM' status codes. The application would almost surely need to be changed to handle this status code correctly.

Handling Unavailable Partitions

© IBM Corporation, 2002

This section describes how application design can handle unavailable partitions.

IMS Version 7

Unavailable Partitions

Causes of unavailable partitions

- Partition stopped in online system
  - Typically, /DBR or /STOP command has been issued for the partition
- Partition authorized to another system without data sharing
  - Typically, batch job is processing the partition
- Partition has flag set in DBRC RECONs
  - Typically, partition needs to be recovered or image copied

© IBM Corporation, 2002

A partition might be unavailable for multiple reasons.

First, the partition might be stopped in the online system. This is typically the result of a /DBR or /STOP command having been issued for the partition.

Second, the partition might be authorized to another system and data sharing is not being used. This system's authorization request would fail. The other system could be a batch job, utility, or online system. Typically, we would encounter this condition when we are running parallel non-data sharing batch jobs.

Third, a flag in DBRC may be set which prevents authorization of the partition. Examples include the 'recovery needed', 'image copy needed', and 'prohibit further authorization' flags.

IMS Version 7

Operational Options

- **Operate as we do with non-HALDB databases**
  - Do not /DBR or /STOP partitions
    - Issue these commands only for the database
  - Do not attempt concurrent access to different partitions from different online systems and batch jobs without data sharing
  - If any database data set is unavailable, stop all access to database

  ➤ Do not have to handle unavailable partitions

  - Use HALDB for:
    - Large databases
    - Shortened database maintenance windows

© IBM Corporation, 2002

We can avoid most unavailable partition conditions by operating as we do with non-HALDB databases.

If we do not issue /DBR and /STOP commands for partitions, we avoid this reason for unavailable partitions. We can issue /DBR and /STOP commands for the HALDB database. This makes the entire database unavailable. This is the same condition that would exist when these commands are used for non-HALDB databases.

If we do not attempt concurrent access to different partitions from different online systems and batch jobs without using data sharing, we avoid another reason for unavailable partitions. Instead, only one system or batch job is allowed to access a HALDB database at any time. This is the same restriction we have with non-HALDB databases. Of course, if we are using data sharing, we can have concurrent access to HALDB databases just as we have for non-HALDB databases with data sharing.

If we treat a problem with a partition as if it were a problem with the entire database, we avoid another reason for unavailable partitions. For example, if we lose a database data set, we can discontinue using the entire database. This is what we would do with a non-HALDB database. For HALDB, we could discontinue using only the affected partition. This would make the rest of the database available, but not this partition. Our application design would have to handle this condition. Instead, we can discontinue using the entire HALDB database. Then, our application design can be the same as it is for non-HALDB databases.

If we use this option, we still have good reasons for using HALDB. We can have very large databases. We can shorten our database maintenance windows in which we do reorganizations, non-fuzzy image copies, and other such activities.

IMS Version 7

Operational Options

- **Operate differently with HALDB databases**
  - Process partitions in parallel by different systems without data sharing
  - /DBR or /STOP partitions
  - If any database data set is unavailable, keep other partitions available
- **Add INIT STATUSGROUP calls**
  - Add to programs to avoid U3303 ABENDs
    - Attempts to access unavailable partitions cause U3303 unless this call is issued
- **Code for 'BA' status code**
  - Must be able to react to different reasons for 'BA'
    - Attempts to access an unavailable partition result in 'BA' status code
    - Other conditions may cause 'BA'
  - React to unavailable partitions, databases, or records

© IBM Corporation, 2002

Our other option is to take further advantage of HALDB. In this case, we can process partitions in parallel with different non-data sharing systems. We can /DBR or /STOP individual partitions. If we lose a database data set in one partition, we can continue to use the other partitions.

If we do this, we need to ensure that our applications can handle these potential unavailable partition conditions. They need to have INIT STATUSGROUP calls to avoid 3303 ABENDs. They must be able to correctly handle 'BA' status codes that they receive for unavailable partitions. These are in addition to the other reasons that they might receive 'BA' status codes.

IMS Version 7

DB PCB Status Code Priming

- **When program is scheduled,**
  - DB PCB status code indicates status of database
    - 'blank blank' - database is available
    - 'NA' - database is not available
    - 'NU' - database is not available for update
- **INIT DBQUERY call**
  - Restores the DB PCB status code fields to settings at schedule time
    - Same meanings for 'blank blank', 'NA', and 'NU'
- **No indication of status of partitions**
  - Database may be available, but some or all partitions may not be available
  - Application program does not know which partitions are available

© IBM Corporation, 2002

Before we look at the handling of unavailable partitions, we will look at the priming of the status code field in database PCBs. This does not change with HALDB.

When a program is scheduled the database PCB status code field indicates the status of the database, but not its partitions. A 'blank blank' status code indicates that the database is available. An 'NA' indicates that the database is not available. An 'NU' indicates that the database is available, but not available for update.

The INIT DBQUERY call may be used to get the same information in the status code fields. That is, it restores these fields to the settings that they had when the application was scheduled.

The important thing to note is that there is no information about the status of individual partitions. A database may be available, but some or all of its partitions may not be available. This means that a 'blank blank' status code field does not mean that the partition you might want to access is available. The program is not given a way to determine the status of partitions.

IMS Version 7

Reasons for 'BA' Status Code

- **Unavailable partition or database**
  - Stopped due to /DBR, /STOP, or /LOCK command
  - Not available for update due to /DBD or access intent of RD or RO
  - DBRC authorization failed
    - Authorized to another subsystem which is incompatible with this subsystem or
    - Flag set in RECONs (Prohibit further authorization, IC Needed, ...)
- **Unavailable record**
  - Lock held by failed IMS subsystem
  - Lock held by in-doubt UOW in failed commit manager
    - CICS, DB2 Stored Procedure, other ODBA connector

© IBM Corporation, 2002

If we issue an INIT STATUSGROUP call, we may receive a 'BA' status code on a database call. There are multiple reasons that this status code could be received. They come in two categories. The first category is an unavailable partition or database. The second is an unavailable record.

There are multiple reasons that a partition or database might be unavailable. First, a database or partition required for the call has been stopped due to a /DBR, /STOP, or /LOCK command. Second, an update call is issued for a database or partition which cannot be updated. The update is prohibited when a /DBD command has been issued for it or when its access intent is RO or RD. Third, a partition required for the call cannot be authorized. This could be due to a concurrent incompatible authorization to another IMS subsystem or due to a flag setting in the RECONs, such as the "prohibit further authorization" flag or the "image copy needed" flag.

A record may be unavailable to a call because of a lock held for it in a special state. If a lock is held by a failed subsystem, a call requiring the lock will receive the 'BA' status code. If a lock is held by an in-doubt unit of work which is no longer connected to IMS, IMS will return a 'BA' status code for a call requiring the lock. The unit of work could be managed by CICS using DBCTL or by a user of ODBA such as a DB2 stored procedure.

IMS Version 7

Unavailable Data Conditions

How can we know which unavailable data condition exists?

- Partition Not Available

  or

- Record Not Available

If we get a 'BA' status code, what should we do?
Assume an entire partition is not available?
Assume one record is not available?

© IBM Corporation, 2002

Since there are multiple reasons for unavailable data conditions, how can a program determine which is the cause? If only a record is unavailable the program probably needs to do something different than it would do if an entire partition were unavailable. More importantly, how can the program distinguish between reaching the end of a partition and attempting to access the next partition which is unavailable and reaching an unavailable record with many more records available in the partition?

That's what we will address next.

IMS Version 7

Handling a 'BA' Status Code

- **Unqualified GN call after 'BA' for unavailable partition**
  - Retrieves first record in next available partition or gets 'GB' status code
    - Indicates that a partition was unavailable
- **Unqualified GN call after 'BA' for unavailable record**
  - Attempts to access same unavailable record
  - Receives 'BA' status code
    - Indicates that a record was unavailable

> **Potential problem:**
> - If first call gets 'BA' due to unavailable partition and
> - First record in next available partition is unavailable (lock reject)
> - Second call (unqualified GN) will get 'BA'

© IBM Corporation, 2002

IMS has special processing for certain calls after a 'BA' status code is received. When the 'BA' is due to an unavailable partition and the next call using the PCB is an unqualified GN call, IMS retrieves the first record in the next available partition. If there are no more available partitions, a 'GB' status code is returned for the GN call. One could use this characteristic to understand if the 'BA' was due to an unavailable partition.

If the 'BA' was due to an unavailable record, that is, one of the lock conditions, a subsequent unqualified GN call would attempt to access the same record. It would again receive the 'BA' status code. One could use this characteristic to assume that the 'BA' was due to an unavailable record.

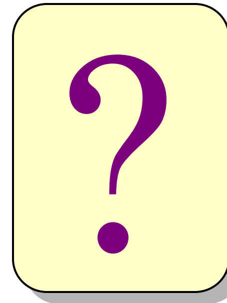
There is a potential problem with this logic. If the first call gets a 'BA' because the partition is unavailable, a subsequent unqualified GN call could get also get a 'BA' code. It could be due to an unavailable record condition for the first record in the next available partition. In this case, we might be tempted to think that the two calls tried to access the same unavailable record. That's not true. The first attempted to access an unavailable partition. The second attempted to access an unavailable record.

IMS Version 7

Programming for 'BA'

What should your application do when a partition is unavailable?

- Inform the terminal operator?
  - Is there a terminal operator?
- Quit?
  - How do you restart the process at the right time?
- Skip this partition?
  - What effect does this have on the application?
- Cause this partition to be processed later?
  - How do you implement this?



© IBM Corporation, 2002

If we choose to operate differently with HALDB, programs may encounter unavailable partitions. Of course, the programs should include INIT STATUSGROUP calls. If they do and if they encounter unavailable partitions, what should they do? The answer depends on the application design and the application requirements. Some of the options are listed here.

If there is a real human operator who invokes the transaction, the program could merely reply to the operator that the requested data is not available. This assumes that the program is only invoked by these real operators or that the program can determine that there is one.

The application program could simply terminate. This could be a reasonable option for batch jobs. In this case, the program needs to be run to completion later. The application design would have to include a process to restart the program at the right place when the partition becomes available.

The application program could skip the partition. It could move to the next partition. The application design would have to be insensitive to the missing data or somehow account for it.

If the partition is skipped, the program may need to process it in a later invocation of the program. The application design would have include a process to restart the program and process only this partition.

IMS Version 7

Processing Secondary Indexes as Databases

© IBM Corporation, 2002

This section describes processing of HALDB secondary indexes (PSINDEX) as databases. In some cases, application programs may need to be changed. In some cases, secondary indexes may be defined so that no application programming changes are required.

IMS Version 7

Secondary Indexes

Fields in a secondary index segment:

Search Field	Subsequence Field	Duplicate Data Field	Concatenated Key Field	User Data

- Search field - key of the secondary index segment
- Subsequence field - optional fields to make non-unique keys unique
  - Copied from source segment
    or
  - System-related unique key created by use of '/SX...' field name
  - Concatenated key of source segment created by use of /CK...' field name
- Duplicate data - optional fields copied from source segment
- Concatenated key field - optional field used for symbolic pointing with non-HALDB
- User data - optional fields maintained by user, not IMS
  - Rarely used

© IBM Corporation, 2002

When an application program processes a secondary index as a database, it receives a copy of the secondary index segment in its IO area. This is a review of the fields that may exist in one of these segments.

The search field is the key of the secondary index. It is composed of one or more fields from the indexed database source segment.

Subsequence fields are optional. They are used to make the VSAM KSDS key unique. The KSDS key is composed of the search field and the subsequence field. The search field may not be unique. To make KSDS keys unique, the database designer uses subsequence fields. There are three types of subsequence fields. First, they may be fields that exist in the source segment. Second, they may be system generated. These fields use "/SX" as the beginning of their names. For non-HALDB indexes, system generated fields are the relative byte address (RBA) of the source segment. Since all RBAs are unique, they are unique. For HALDB indexes, system generated fields are the Indirect List Key (ILK) of the source segment. Since all ILKs are unique, they are unique. The third type of subsequence field is a concatenated key field. It is the concatenated key of the source segment. These fields use "/CK" as the beginning of their names.

Duplicate data fields are optional. They are not part of the key of the index. They are fields that are copied for the source segment. They are used to add data to a secondary index so this data will be available to programs which process the secondary index as a database.

Concatenated key fields are optional. They are used for symbolic pointing with non-HALDB secondary indexes. When symbolic pointing is used, the pointer is not in the prefix. Instead, it is this concatenated key field. HALDB never uses symbolic pointing.

User data fields are optional. They are fields that are maintained by users, not by IMS. These fields are updated by application programs which process the secondary index as a database. User data fields are rarely used. When a non-HALDB database is reorganized, secondary indexes with direct pointers must be rebuilt. This is done with utilities. These utilities do not recreate the user data. This restriction limits the usefulness of user data fields.

IMS Version 7

Processing a Secondary Index as a Database

- **Subsequence field size increased when using /SX**
  - /SX field increased from 4 to 8 bytes for HALDB
    - KEYLEN in PCB must be increased by 4 bytes
    - Duplicate data fields and user data are offset by 4 bytes
      - IO-area must be adjusted

Program I/O Area for Non-HALDB Secondary Index Segment

Search Field	Subsequence Field with /SX	Duplicate Data Field	User Data

Program I/O Area for HALDB Secondary Index Segment

Search Field	Subsequence Field with /SX	Duplicate Data Field	User Data

© IBM Corporation, 2002

The /SX system related field contains an 8 byte ILK for HALDB secondary indexes. It contains a 4 byte RBA for non-HALDB secondary indexes. This changes the key size and the offset to the duplicate data part of the segment.

The KEYLEN value in PCBs which reference this secondary index as a database must be at least as large as the new key size. If the old value for KEYLEN accurately reflected the actual key size, it must be increased by 4 bytes.

Since the subsequence field is now 4 bytes larger, the offset to any duplicate data or user data fields must be adjusted by 4 bytes. Also, the IO area used by the application program must be large enough to hold the larger segment.

An index may have multiple subsequence field sources. This means that one or more fields from the source segment may also be used in the subsequence field. Typically, this is not done with /SX fields since /SX fields always guarantee uniqueness. But, if these additional fields are defined and if they follow the /SX field, the offset to these fields will also change by 4 bytes.

IMS Version 7

Are You Affected by /SX?

- **If you have PSB which specifies DBDNAME as a secondary index name:**

```
PCB   TYPE=DB,DBDNAME='sec. index name',KEYLEN=n
```

- **Find LCHILD in the DBD for this secondary index:**

```
LCHILD NAME=(segmentname,databasename),INDEX=xdfldname
```

- **Match INDEX value from secondary index LCHILD with NAME value from XDFLD statement in indexed database:**

```
XDFLD NAME=xdfldname,SRCH=list,SUBSEQ=/SXzzzzz,DDATA=list
```

- **If "/SX" appears in SUBSEQ value, KEYLEN must be increased in PSB and size of IO area must be increased**

- **If DDATA is also specified, reference to dup. data fields must be adjusted**

© IBM Corporation, 2002

This page explains how to determine if your program is affected by the increased size of /SX fields.

First, examine your PCBs. If the DBDNAME field on a PCB refers to a secondary index, you may be affected.

If so, examine the LCHILD statement in the DBD for the secondary index (PSINDEX). Match the name specified in the INDEX parameter for this LCHILD statement with NAME parameter on an XDFLD statement in the indexed database (PHDAM or PHIDAM). Examine this XDFLD statement. If its SUBSEQ value includes a field name beginning with "/SX", this secondary index is using a system generated field. In this case, the key length of the segment is 4 bytes larger than it was as a non-HALDB secondary index. This means that the KEYLEN field on the PCB and the size of the IO area must be large enough to handle the increased key size and the increased segment size. Next, you must determine if there are any duplicate data fields residing past (to the right of) the subsequence field. If their is a DDATA parameter on this XDFLD statement, these duplicate data fields exist. Application program offsets to these fields must be adjusted for the extra 4 bytes.

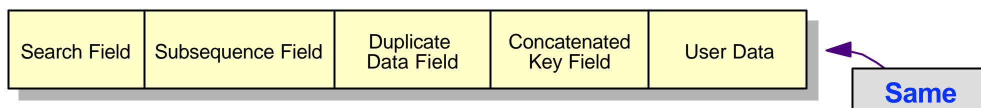
If you have user data fields in this segment, the offsets to them must also be adjusted for the extra 4 bytes. User fields are not directly defined in the secondary index DBD. Instead, they exist if the segment size is larger than that required to hold the other fields (Search, Subsequence, etc.) of the segment.

IMS Version 7

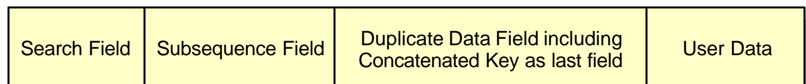
Processing a Secondary Index as a Database

- **Symbolic pointing not used with HALDB**
  - Concatenated key field not present
    - Application may not react correctly
- **Solution:**
  - Concatenated key may be retained as duplicate data field
    - No changes required to application programs

Program I/O Area for Non-HALDB Sec. Index Segment using Symbolic Pointing



Program I/O Area for HALDB Sec. Index Segment with Concat. Key as Duplicate Data



© IBM Corporation, 2002

As was mentioned at the beginning of this section, HALDB does not use symbolic pointing. When a non-HALDB secondary index using symbolic pointing is migrated to HALDB, the concatenated key field will no longer be present. It appears that application programs which process the secondary index as a database and which use the concatenated key field or user data fields, would have to be adjusted. But, there is a simple solution for this.

The definition of the HALDB secondary index (PSINDEX) can be used to maintain the same segment from the application point of view. This is done by defining a new duplicate data field in the PSINDEX. This duplicate data field is for the concatenated key. By placing it at the end of the duplicate data, it occupies the place where the concatenate key field appeared in the non-HALDB secondary index segment.

IMS Version 7

Are You Affected by Symbolic Pointing?

- If you have PSB which specifies DBDNAME as a secondary index name:

```
PCB   TYPE=DB,DBDNAME='sec. index name',KEYLEN=n
```

- Find LCHILD in the DBD for this secondary index:

```
LCHILD NAME=(segmentname,databasename),PTR=SYMB
```

- If "PTR=SYMB" appears on LCHILD, you have symbolic pointing
- Keep concatenated key in I/O area by adding "/CKxxxxx" field to DDATA

```
XDFLD NAME=xdfldname,SRCH=list,DDATA=(list,/CKxxxxx)
```

  - /CKxxxxx field must also be defined with FIELD statement in indexed database

© IBM Corporation, 2002

This page explains how to determine if your program is affected by the elimination of symbolic pointing.

First, examine your PCBs. If the DBDNAME field on a PCB refers to a secondary index, you may be affected.

If so, examine the LCHILD statement in the DBD for the non-HALDB secondary index. If it has PTR=SYMB, it is using symbolic pointing.

If your application program uses the concatenated key field or user data field, you should add the concatenated key field as a duplicate data field in the HALDB secondary index (PSINDEX). This is done by including a field beginning with "/CK" as the last or only field in the DDATA list for the XDFLD statement in the indexed database. This "/CK" field must also be defined in a FIELD statement for the source segment in the indexed database.

Converting from User Partitioning

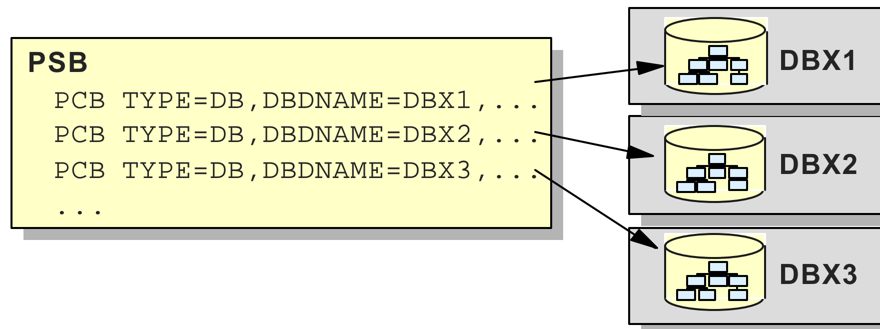
© IBM Corporation, 2002

This section describes application considerations for the migration of user partitioned databases to HALDB.

IMS Version 7

User Partitioning

- Some installations have done their own partitioning
  - Database split into multiple databases
  - Application selects which database to use
    - Based on key of root segment
    - Database selection may be done by subroutine or modification to language interface module
    - PSB has PCB for each database



© IBM Corporation, 2002

Some installations have done their own partitioning. Typically, they did this to address database size limitations many years ago. This technique is called "user partitioning" because it does not use a product such as IMS/ESA Partition Support. Instead, the installation creates multiple IMS databases. Each database contains a portion of the records that the application wants to view as one database. Since there are multiple databases, each requires its own PCB in PSBs. The application has to select the proper PCB for the record with which it is dealing. The application can make this selection in several ways. Typically, one of two ways is used. First, the program may invoke a subroutine. The key of the record is passed to the subroutine. The subroutine selects the proper PCB and makes the call. Second, the program may use a modified version of the IMS language interface module. With this scheme, the program always references the first PCB for the user partitioned database. The module selects the proper PCB and modifies the call to use it.

In this example, database DBX has been split into three IMS databases, DBX1, DBX2, and DBX3. The PSB includes PCBs for each of these databases.

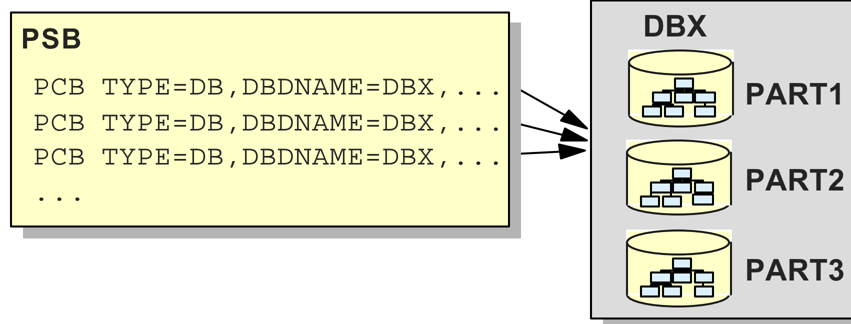
Obviously, user partitioned databases are prime candidates for migration to HALDB. When they are migrated, they become a single HALDB database with multiple partitions.

IMS Version 7

User Partitioning

Converting from user partitioning

- Multiple databases become one HALDB database with multiple partitions
- Application does not need to select which database to use
  - Could use one PCB - would require application changes
- Alternative:
  - PSB is changed so that all PCBs reference the same HALDB database
  - Application program continues to select PCBs - no application changes



© IBM Corporation, 2002

When a user partitioned database is migrated to HALDB, we might think we need to eliminate all but one of our PCBs from our PSBs. It turns out that this is not required. Instead, we can merely change all of the existing PCBs to reference the same HALDB database. This could make the migration simpler. We do not have to make other modifications to the application program. For example, we can continue to use a subroutine which selects a PCB. No matter which PCB it selects, it will be correct. Each PCB points to the same HALDB database. If we use a modified language interface module, we can continue to use it.

Of course, we will probably want to eliminate our PCB selection routine or our modified language interface module at some time. The point of this technique is that this work is not required as part of our migration to HALDB.

IMS Version 7

Summary

- **Partitioning Options**
  - Key range vs. partition selection exit routine
- **Initial Loads**
  - Current programs work except for loading logical children
  - Can load partitions in parallel
- **Processing Partitions in Parallel**
  - Use HALDB control statement to limit a PCB to a partition
- **Handling Unavailable Partitions**
  - Optional, could operate as today without making individual partitions unavailable
  - Understand 'BA' status code use
- **Processing Secondary Indexes as Databases**
  - May require changes to application programs, especially with duplicate data and /SX field
  - May require changes to secondary index definition for symbolic pointer fields
- **Converting from User Partitioning**
  - PSB change will probably be sufficient

© IBM Corporation, 2002

This presentation has covered many topics concerning HALDB database and application design.

During the introduction the partitioning options were explained. Application requirements may cause you to choose a particular scheme for partitioning.

We looked at initial loads of HALDB databases. This included the required order of records when loading a database, how to load partitions in parallel, and how to handle logical children.

We looked at processing partitions in parallel and described how the HALDB control statement may be used to limit a PCB to one partition. We also saw the limitation of this approach when secondary indexes or logical relationships are used.

We looked at the handling of unavailable partitions. We saw that we can continue to operate as we do today without making individual partitions unavailable. If we do make partitions unavailable, we will probably need to modify programs to handle the 'BA' status code correctly.

We saw how secondary indexes which are processed as databases are affected by migration to HALDB.

Finally, we looked at the conversion of user partitioned databases to HALDB.