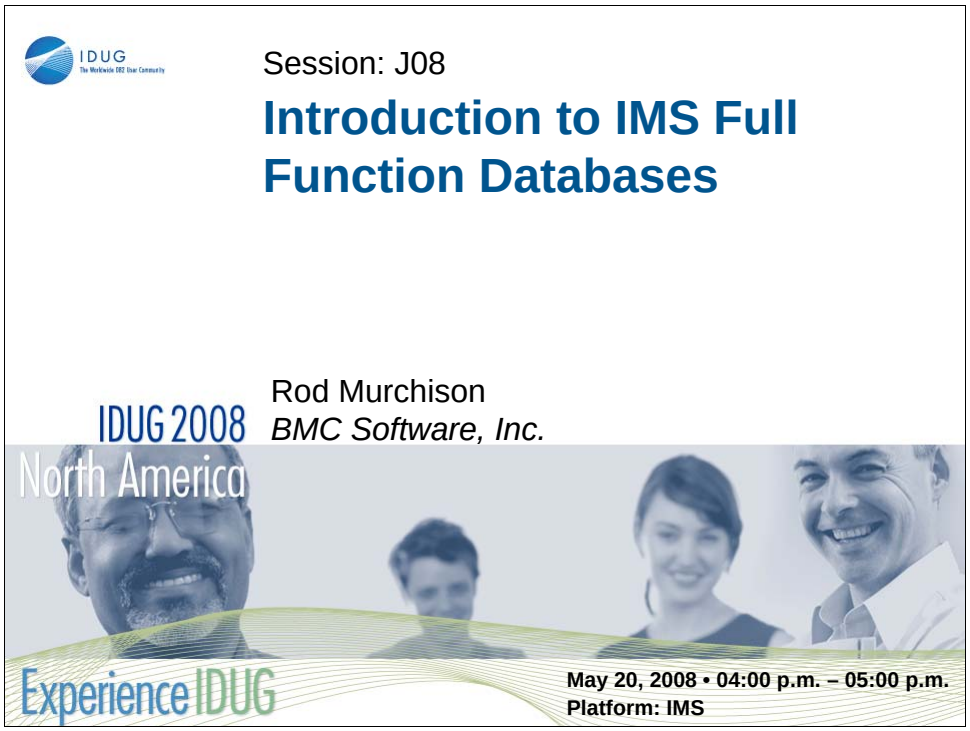


This session provides an introduction to IMS full function databases. It explains the strengths and weaknesses of the various types of database and gives some guidance on when to use each type. The way in which data is stored, manipulated, and retrieved in each of the database organizations is covered. The information is given at an introductory level and is meant for people who want a high-level understanding of IMS database concepts.

Objectives

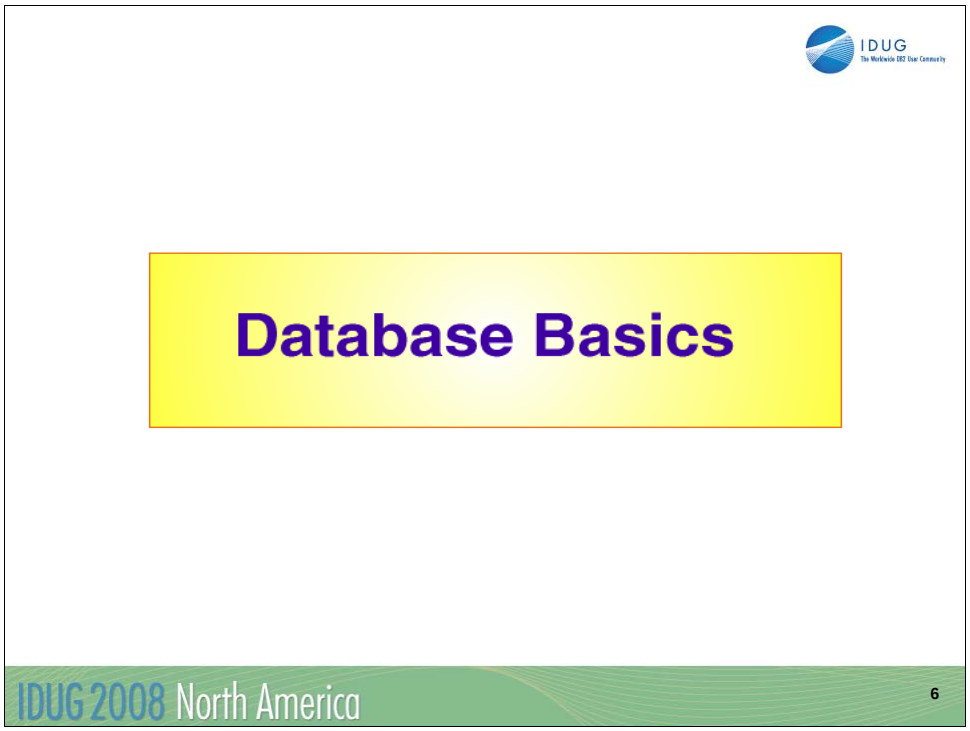
- Understand the structure and terminology of IMS databases
- Be able to select the database organization that is most appropriate for an application
- Understand how data is stored and manipulated in each of the database organizations
- Know how direct pointers are used and maintained
- Understand how free space is used in direct DBs

Contents

- Database Basics
  - What is a Database
  - The IMS Database
  - The Hierarchy
  - Segment Rules
  - Segment Relationships
  - Hierarchic Sequence
  - Access to Segments
  - Segments in Storage

Contents…

Contents…


IDUG
Database Basics
IDUG 2008 North America
6

What is a Database

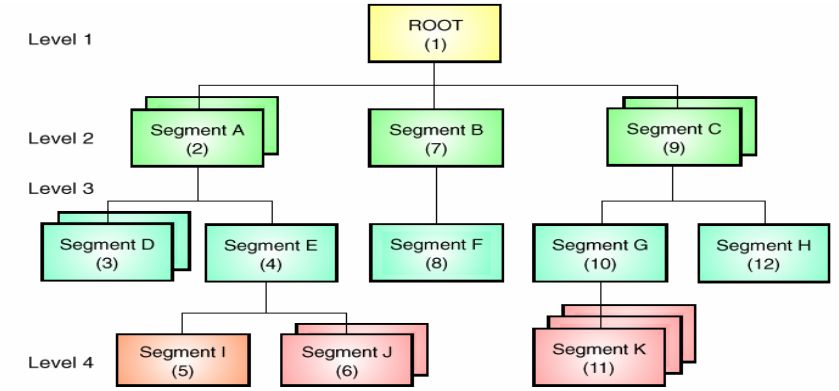
- A collection of interrelated data items organized in a form for easy retrieval
  - The collection of data is stored in a computer system
  - The retrieval is done by application programs
  - Each item of data only needs to be stored once
    - Shared among programs and users
- An IMS database is organized as a Hierarchy
  - Levels of data
  - Data at lower levels depends on data at higher levels for its context
    - You cannot understand the lower level without knowing the higher levels

The IMS Database

- A database is a collection of related database records
- A database record is a single hierarchy of related segments
- A segment is a group of related fields
- A field is a single piece of data
  - It can be used as a key for ordering the segments
  - It can be used as a qualifier for searching
  - It may only have meaning to the applications

- IMS databases always look like hierarchies

The Hierarchy

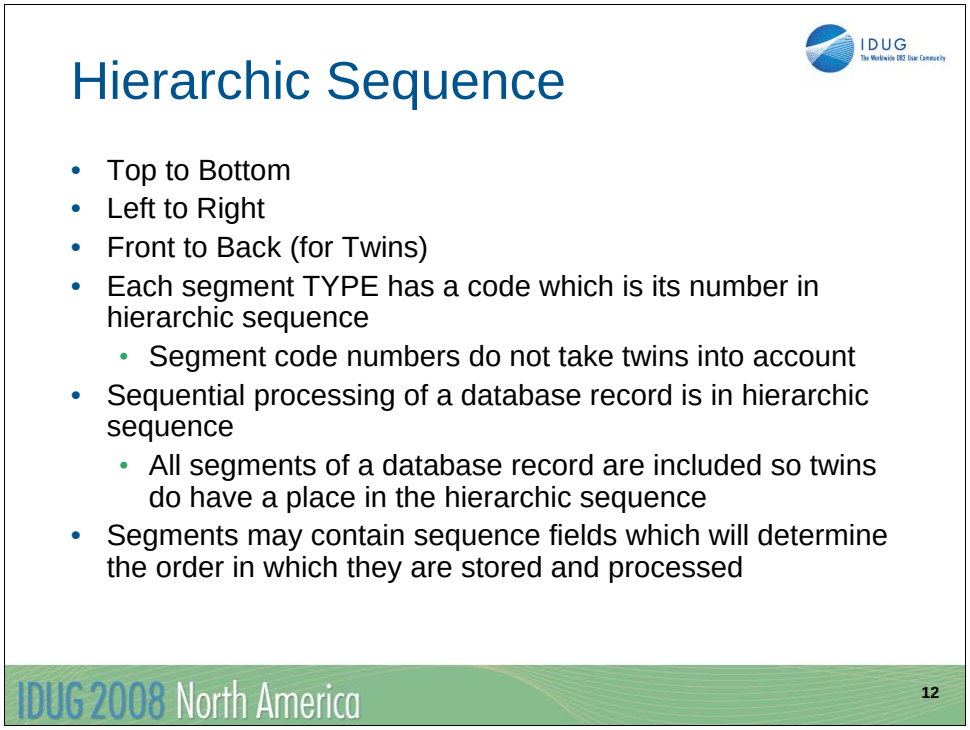


Segment Rules

- Root
  - One and only one for each database record
  - No higher level segments
    - Everything depends on the information in the root
- Other segment types
  - Up to 254 different segment types
    - 255 including the root
  - Any number of occurrences of each segment type
  - Each segment, except the root, is related to one and only one segment at the next higher level
- Levels
  - Up to 15 levels

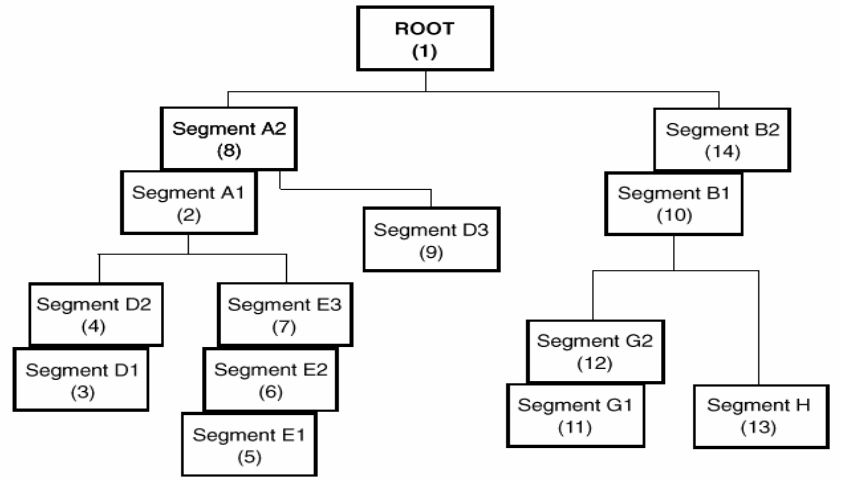
Segment Relationship

- Parent
  - Any segments which has dependents at the next lower level is the parent of those segments
  - A parent may have any number of dependent segments
- Child
  - Each segment which depends on a segment at a higher level is a child of that segment
  - Each child segment has one and only one parent
- Twin
  - All occurrences of the **same** segment type under the **same** parent are twins
  - There may be any number of twins
- Siblings
  - Segments of **different** types with the **same** parent are siblings



The maximum of 255 different segment TYPES means that the segment code can be stored as a single byte. When determining the segment type code, you go through the segment types in hierarchical sequence. When IMS processes the database sequentially, it will go through all of the segment occurrences in hierarchic sequence.

Hierarchic Sequence…



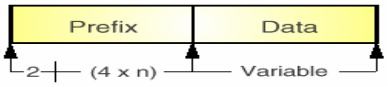
Access to Segments

- Retrieval
  - Get Unique (GU)
    - Read a particular segment, by sequence or search fields
  - Get Next (GN)
    - Read the next segment in hierarchic sequence
  - Get Next Within Parent (GNP)
    - Read the next segment in hierarchic sequence under a particular parent

Access to Segments…

- Update
  - Insert (ISRT)
    - Adds a new occurrence of a segment to the database
  - Delete (DLET)
    - Delete an occurrence of a segment
  - Replace (REPL)
    - Update a segment with new data, not the sequence field

Segments in Storage



- **Segments are stored with a prefix and a data portion**
  - The prefix is used only by IMS
  - The data is what the application program sees

- **The prefix contains :**
  - SC = segment code, 1 byte
  - DB = delete byte, 1 byte
  - 0 to n pointers, 4 bytes each

IDUG
The Worldwide DB2 User Community

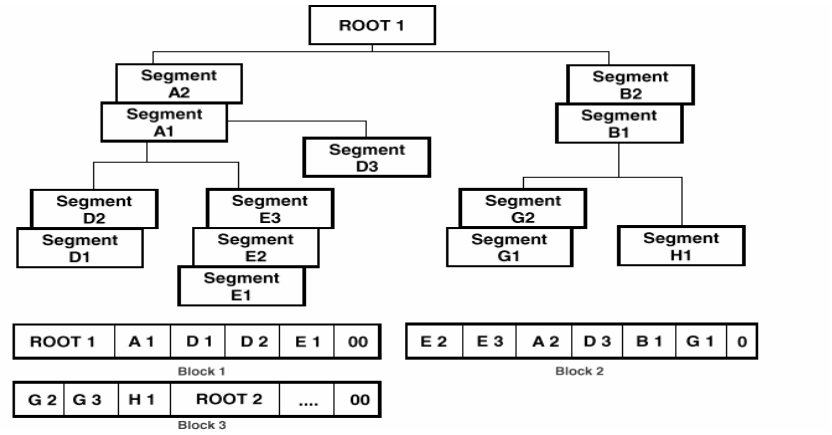
Sequential Organization

- **The data is physically stored in hierarchic sequence**
  - Database records are stored in root key sequence
    - If no root key, they are stored as presented
  - Segments in a record are stored in hierarchic sequence
- **Sequential Database Types**
  - Hierarchic Sequential Access Method (HSAM)
  - Simple Hierarchic Sequential Access Method (SHSAM)
    - Root-only HSAM
  - Hierarchic Indexed Sequential Access Method (HISAM)
  - Simple Hierarchic Indexed Sequential Access Method (SHISAM)
    - Root-only HISAM using VSAM
  - Generalized Sequential Access Method (GSAM)
    - No hierarchy, no database records, no segments

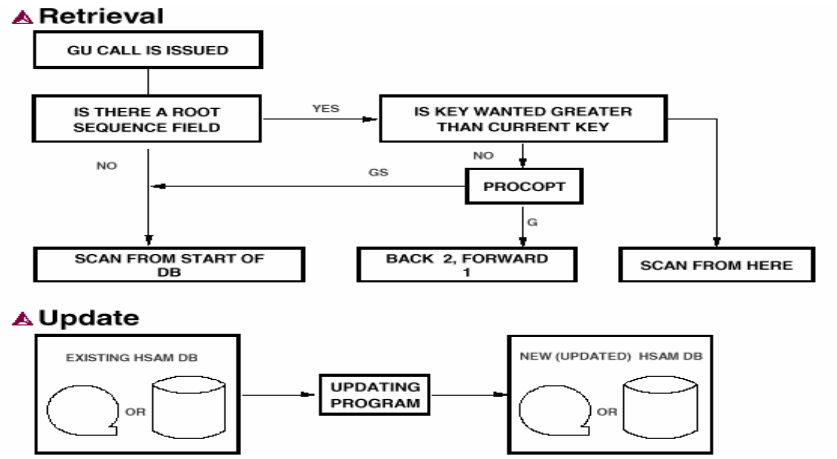
HSAM

- Tape or DASD
  - Only from that can use tape
- Fixed-length records, Unblocked format
  - RECFM=F, logical record length=block size
- Cannot Delete or Replace
  - Update is by rewriting the database
- Insert allowed only when loading the database
- Restrictions
  - No pointers in the prefix, SC and DB only
    - Delete byte is not used
  - No multiple data set groups (MDSG)
  - No logical relationships or secondary indices
  - No variable length segments
  - No segment edit/compression or data capture
  - No logging, recovery, or reorganization

HSAM Storage



HSAM Processing



SHSAM

- HSAM with only one segment type (Root-only)
  - No prefix is used
    - No SC because only one segment type
    - No DB because HSAM does not use it anyway
- Same restrictions and processing as HSAM
- Fully equivalent to a standard QSAM or BSAM file
  - Communication with non-IMS systems
  - Passing very large volumes of data

HISAM

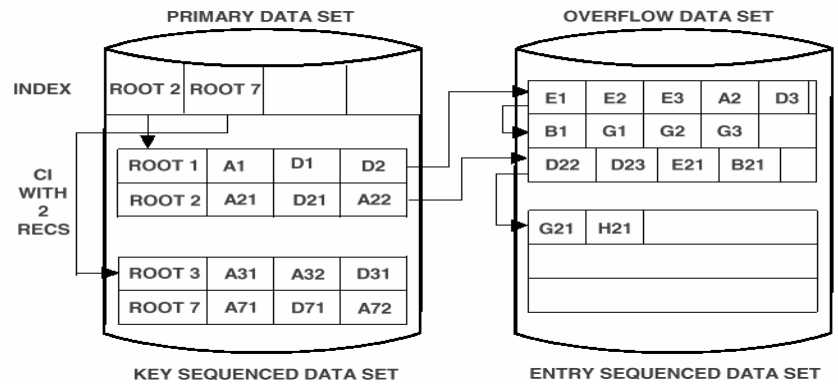
- DASD only
- VSAM
  - KSDS for the primary data set
  - ESDS for the overflow data set
- Each root must have a unique key
- A database record is stored as 1 record in the primary data set and 0 to N records in the overflow dat set
- All calls are allowed
- Prefix consists of SC and DB
- HSAM restriction do not apply

HISAM…

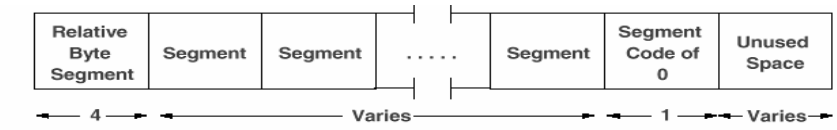
HISAM works better when

- Application randomly access the records and then read the segments sequentially
- Most of the database records are the same size
- Relatively few dependents per root
- Very low insert / delete activity

HISAM Storage



HISAM VSAM Logical Record

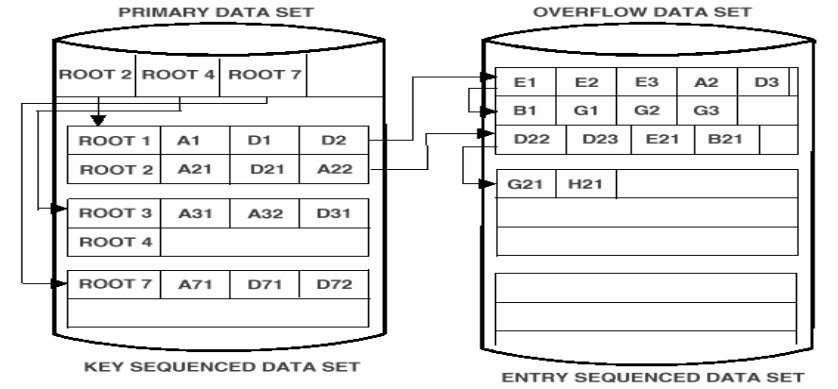


- **RBA pointer to the next logical record for this database record**
  - Last logical record for DB record has zeros
- **Segments are stored in hierarchic sequence**
- **SC of zero indicates end of segments in this logical record**
- **Unused space can have any data in it**

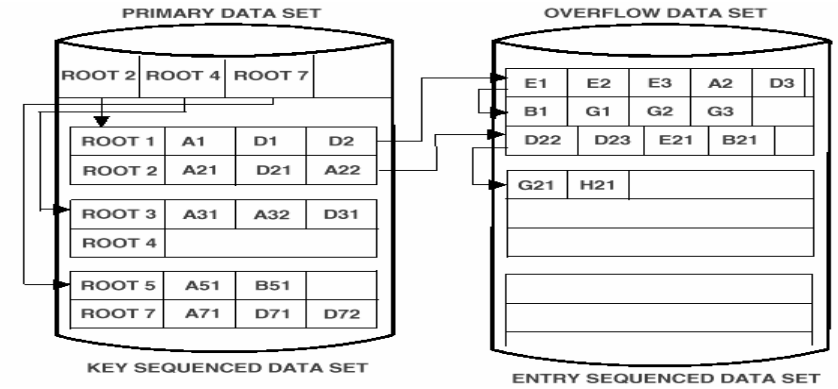
HISAM Inserts

- **HISAM Roots are always inserted into the Primary Data Set (KSDS)**
  - 1. If there is an free record in the VSAM Control Interval (CI)
    - Inserted in root key sequence
    - Higher keys are 'pushed down' to make space
  - 2. If there is no free record in the CI
    - CI is split - some of the records moved to a new CI
      Split at midpoint or insert point by INSERT = in DFSVSAMP
    - After split, same as free record case
- **Dependents are inserted in their place in hierarchic sequence**
  - 1. If there is room in the logical record
    - Following are 'pushed down' to make space
  - 2. If there is not enough room
    - All following segments are moved to a new overflow record
    - Overflow records chain is updated
    - Segment is inserted

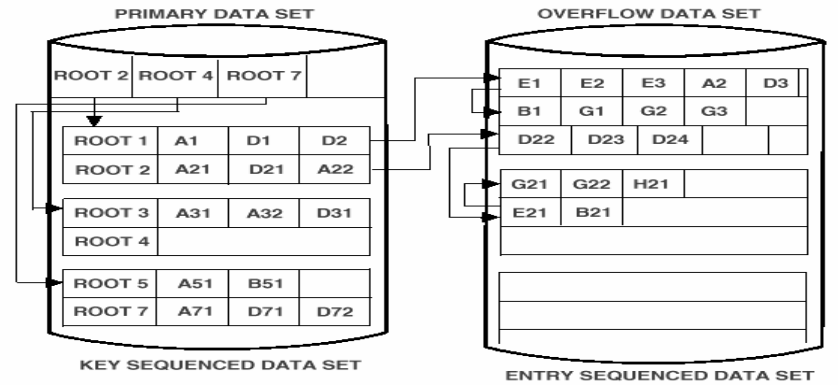
Insert Root 4



Insert Root 5



Insert Dependents G22 and D24



HISAM Delete and Replace

Delete

- Marked as deleted in the Delete Byte in prefix
  - Dependents are not flagged but can't be accessed
- Continue to take up space
  - Unload / Reload to reclaim space
- If the root is deleted and no logical relationship exists
  - The record is deleted from the primary data set
  - Overflow records continue to exist in the overflow

Replace

- Fixed length or same length
  - Overwrite previous data
- Variable length
  - Other segments in the record move to make space
  - Displaced segments will go to a new overflow record

SHISAM

- **HISAM with only one segment type (root-only)**
  - No prefix is used
    - No SC because only one segment type
    - No DB because logical record is deleted
- **Restrictions**
  - No logical relationships or secondary indices
  - No multiple data set groups
  - No variable length segments
  - No edit / compression
- **Fully equivalent to a VSAM KSDS**
  - No ESDS because no dependent overflow
  - Can be accessed by native VSAM programs

GSAM

- **Compatible with MVS data sets**
  - No hierarchy
  - No database records
  - No segments and no keys
- **GSAM VSAM**
  - ESDS on DASD
  - Fixed or variable length records
- **GSAM QSAM / BSAM**
  - Physical sequential (DSORG = PS) on DASD or Tape
  - Fixed, variable, or undefined length records
- **GSAM Processing**
  - No Delete or Replace
  - Insert only at the end of the data set
  - Gets by sequential scan

GSAM…

Restrictions

- No multiple data set groups
- No logical relationships or secondary indices
- No edit / compression or data capture
- No field level sensitivity
- No logging or reorganization

Checkpoint and Restart

- IMS symbolic checkpoint supports GSAM
- Can restart from checkpoint instead of reprocessing
- Restart repositions in the GSAM data set

IDUG
The Worldwide DB2 User Community

TOPIC

Direct Organization

Direct Organization

- Physical storage is independent of hierarchic sequence
  - Pointers are used to maintain segment relationships
    - Pointers are in the segment prefix
    - Segments can be stored ‘anywhere’
    - Segments are not physically moved
  - Space from deleted segments can be reused
- Direct Database Types
  - Hierarchic Direct Access Method (HDAM)
    - Uses a randomizer for direct access to root segments
  - Hierarchic Indexed Direct Access Method (HIDAM)
    - Uses an index to locate root segments

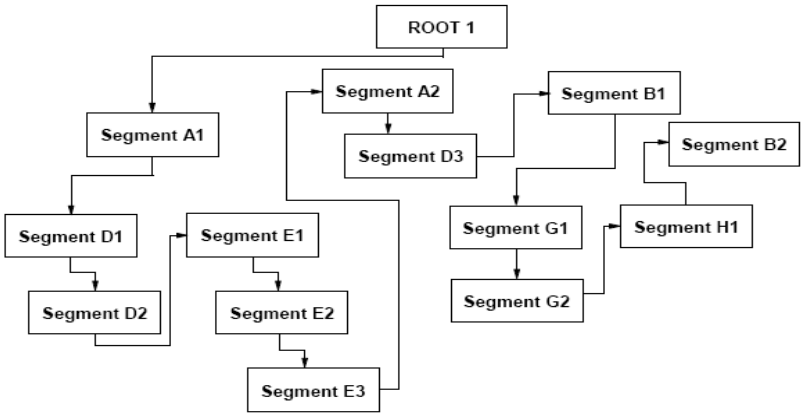
Pointer Types

- Hierarchic
  - May be present in all segment types
  - Hierarchic Forward (HF)
    - Points to the next segment in hierarchic sequence
  - Hierarchic Backward (HB)
    - Points to the previous segment in hierarchic sequence
    - Must also have HF pointers
- Physical Child
  - Found only in the prefix of a parent segment
  - Physical Child First (PCF)
    - Points to the first occurrence of a child segment type
  - Physical Child Last (PCL)
    - Points to the last occurrence of a child segment type
    - Must also have a PCF pointer

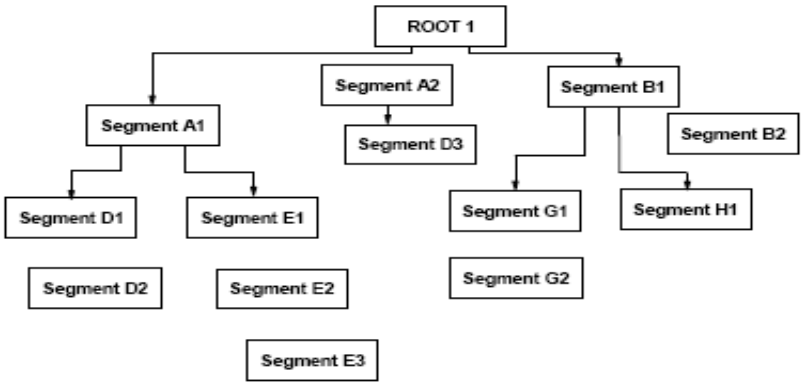
Pointer Types…

- Physical Twin
  - Physical Twin Forward (PTF)
    - Points to the next twin in key or entry sequence
  - Physical Twin Backward
    - Points to the previous twin
    - Must also have a PTF pointer

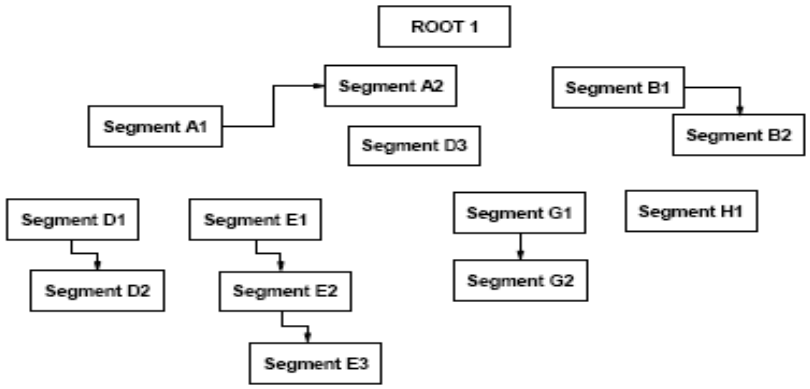
Hierarchic Forward Pointers



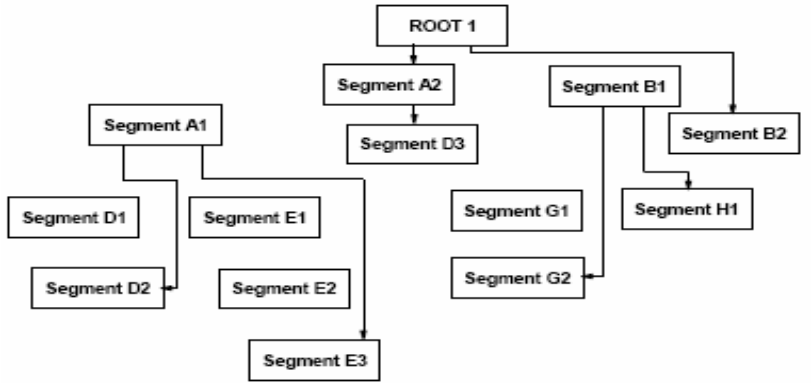
Physical Child First Pointers



Physical Twin Pointers



Physical Child Last Pointers



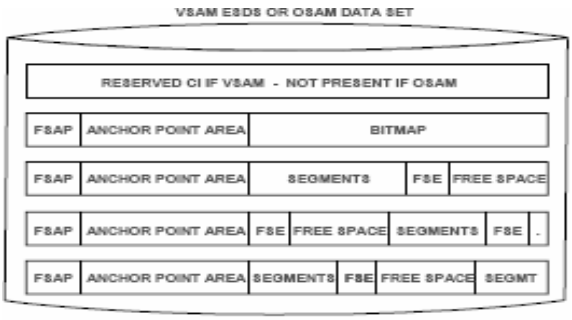
Pointer Uses

- Hierarchic Forward
  - Primary processing is in hierarchic sequence
- Hierarchic Backward
  - Delete activity via a logical relationship or secondary index
- Physical Child First
  - Random processing
  - Sequence field is used **or** insert rule of FIRST or HERE
- Physical Child Last
  - No sequence field **and** insert rule LAST
  - Use of *L command code
- Physical Twin Forward
  - Random processing
  - Needed for HDAM roots
  - Poor choice for HIDAM roots
- Physical Twin Backward
  - Improves delete performance
  - Processing HIDAM roots in key sequence

Pointers in the Prefix

- Cannot have Hierarchic and Physical in the same prefix
  - PTR=H will cause PCF specification to be **ignored**
- **If the parent has PTR=H, the children cannot have backward pointers**
- **If the parent has PTF=HB, the children must have backward pointers**
- **Child pointers will behave according to the parent specification**
  - Parent Hierarchic, last twin points to sibling instead of being 0
  - Parent Twin. Last hierarchic pointer in twins is 0

HD Storage



- **All HD data is in a single ESDS or OSAM data set**
- **The logical records are unblocked**
  - Logical record length = block size for OSAM
  - Logical record length = block size –7 for VSAM
- **All segments stored as an even number of bytes**

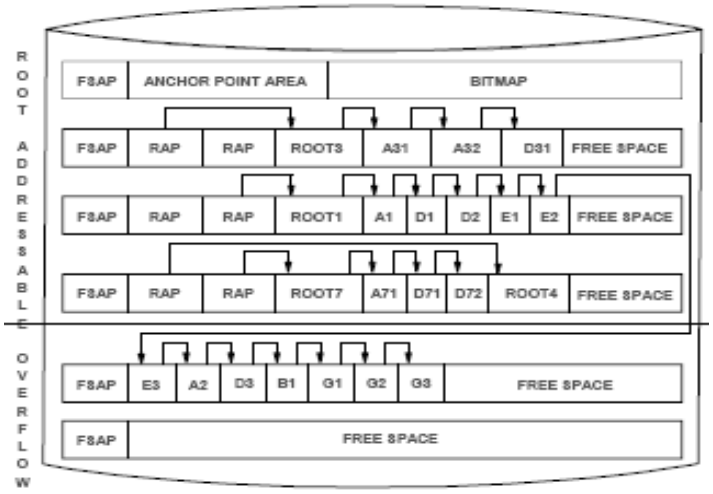
Special HD Fields

- Bitmap
  - One per block or CI
    - First bit represents the bitmap itself
  - 1 = enough space to store the **LONGEST** segment in the database
  - 0 = not enough space for the **LONGEST** segment
  - If a bitmap has N bits, block or CI N+1 is a new bitmap
- Free Space Anchor Point (FSAP)
  - Two 2-byte fields
    - First is the offset in bytes to the first FSE
    - Second, is a flag indicating if this is a bitmap (0 = NOT a bitmap)
- Anchor Point Area
  - Contains one or more 4-byte Root Anchor Points (RAP)
    - 1 RAP in HIDAM if the root has PTF or HF pointer
    - RMNAME parameter specifies the number of RAPs for HDAM
  - Each RAP has the address of a root segment or 0

Special HD Fields…

- Free Space Element
  - First 2 bytes are the offset, in bytes, of the next FSE
    - Zero if this is the last FSE in the block or CI
  - Second 2 bytes are the length of the free space, including the FSE
    - No FSE is created if there is less than 8 bytes of free space
  - Last four bytes is the Task ID of the program that freed the space
    - Allows a program to free and reuse the same space without contention

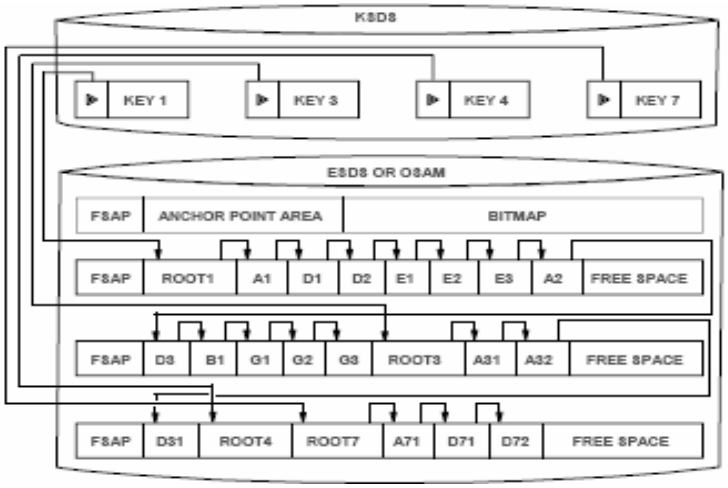
HDAM Storage



HDAM Storage…

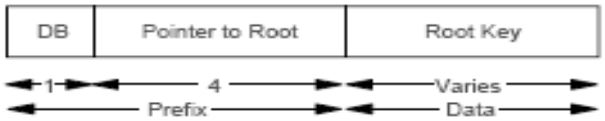
- Root Addressable Area (RAA)
  - Number of blocks or CIs defined in RMNAME parameter
  - Primary storage area for roots and dependents
    - Number of dependents at initial load is limited by RMNAME
    - Insert until specified byte limit would be exceeded
  - All RAPs and in the RAA
  - Location is determined by Randomizer specified in RMNAME
    - Randomizer input is the root segment's key
    - Randomizer output is a block number and a RAP number
    - Keys that randomize to the same block and RAP are synonyms
    - Synonyms are chained using the root's PTF pointer
    - Chain is in ascending key sequence or by insert rules
- Overflow Area
  - For segments that do not fit into the RAA
  - No RAPs are present in the overflow area

HIDAM Storage



HIDAM Storage…

- Data Component
  - A VSAM ESDS or OSAM data set
  - No RAA or overflow area
  - Database records are stored in key sequence when loading
  - Roots must have unique keys
  - Segments are in hierarchic sequence when loaded
  - Can specify free space to be left after loading
- Index Component
  - VSAM KSDS
  - The index is a root-only database
  - One index segment for each database root segment



HIDAM RAP

- One RAP per block or CI if PTR=T or PTR=H for the root
  - No RAP is generated if **backward** pointers are specified
    - Roots are linked in key sequence with backward pointers
  - No RAP is generated if PTR=NOTWIN
- Roots are chained from the RAP in reverse order of insertion
  - RAP points to root most recently inserted in the block or CI
  - Each root points to the previously inserted root
  - First root inserted has a zero pointer
- Index must be used to process roots sequentially
  - Index must also be used if NOTWIN is specified
- Remember that TWIN is the default
  - **Specify** something **useful**
  - Use backward pointers if you process roots in key sequence
  - Use NOTWIN if you do purely random processing

Processing HD databases

- Delete
  - The segment and all of its dependents are removed
  - FSE is used to indicate that the space is free
    - Create a new FSE and update the FSAP/FSE chain
    - Update the length of previos FSE if it follows existing free space
  - Segment pointers are updated to skip the deleted segment
- Replace
  - Fixed-length or no change in length
    - Old segment is overwritten with new data
  - Shorter segment
    - Space previously occupied is freed
    - FSE created if at least 8 bytes shorter
  - Longer segment
    - If adjacent free space permits, store in original location
    - If no space available, then separate data
      - Data portion of segment goes in overflow with prefix of SC and DB='FF'
      - Prefix stays where it is and bit 4 of DB is set on (1)
      - Pointer to data follows prefix and remainder is freed

Processing HD Databases…

- Insert
  - Store in the Most Desirable Block (MDB)
    - HDAM root MDB
      - The one selected by the randomizer
      - The one containing its previous synonym
    - HIDAM root MDB
      - If no backward pointer, same block as root with next higher key
      - If backward pointer, same block as root with next lower key
    - Dependents
      - If Physical pointers, same block as parent or previous twin
      - If Hierarchic pointers, same block as previous segment in hierarchic sequence

Session J08

Introduction to IMS Full Function Databases

Rod Murchison

BMC Software, Inc.

rod_murchison@bmc.com