ODYSSEUS/EduCOSMOS Project #3: EduBtM Project Manual

Version 1.0

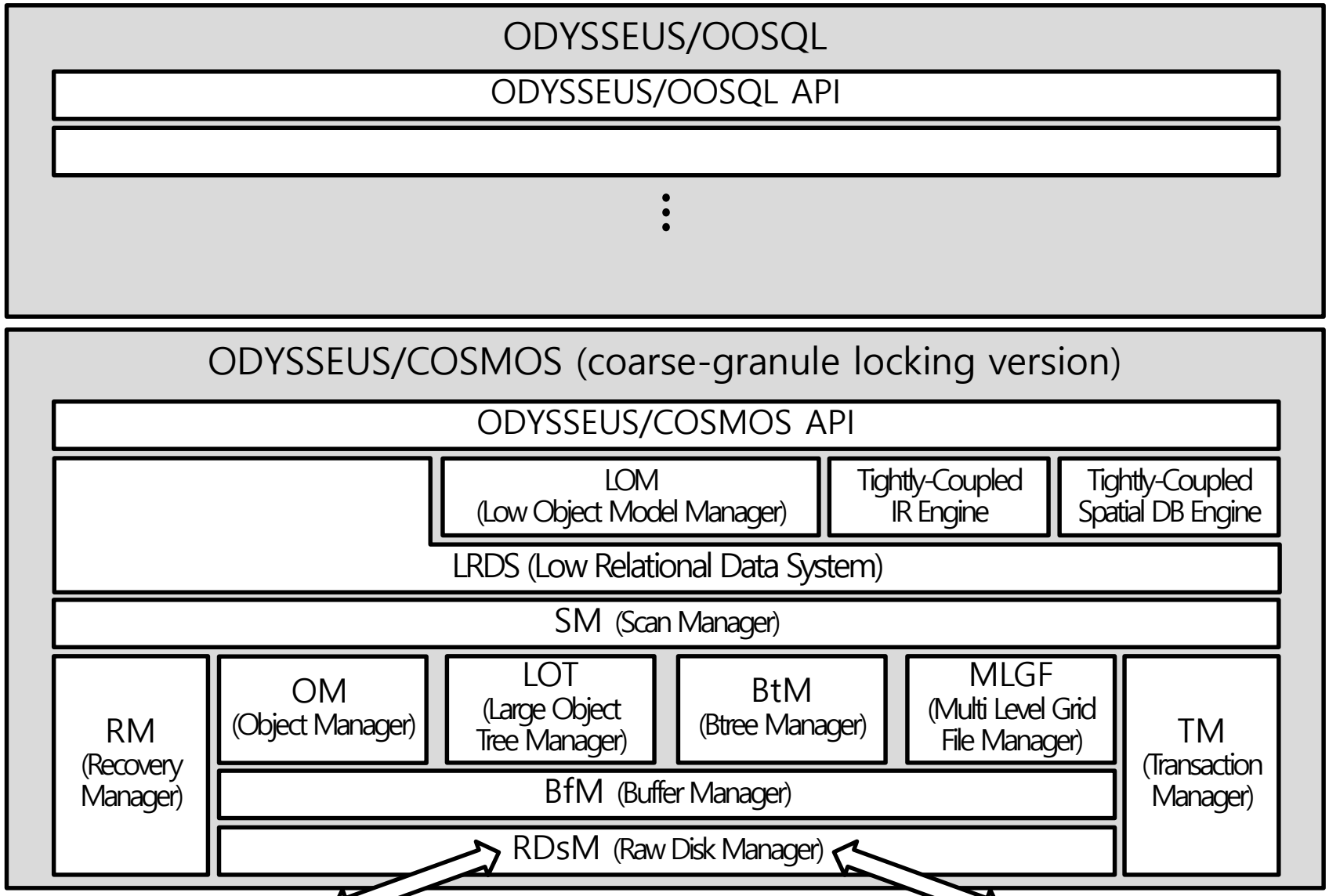
Contents

ODYSSEUS/COSMOS

- ODYSSEUS
  - An object-relational DBMS developed by Kyu-Young Whang et al. at Advanced Information Technology Research Center (AITrc) / Computer Science Department of KAIST. ODYSSEUS has been being developed since 1990.

- ODYSSEUS/COSMOS
  - The storage system of ODYSSEUS, which is used as an infrastructure for various database application softwares.

- ODYSSEUS architecture

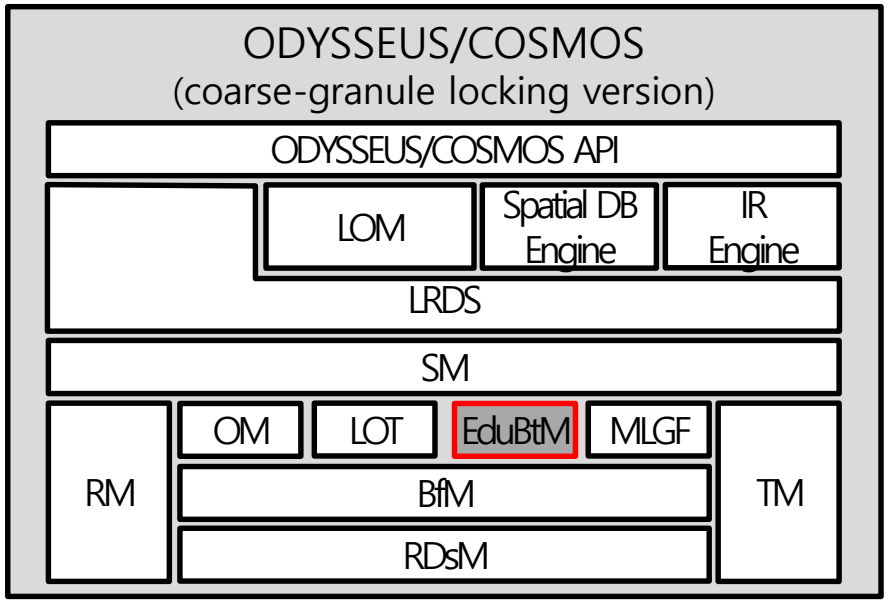


ODYSSEUS/EduCOSMOS Project

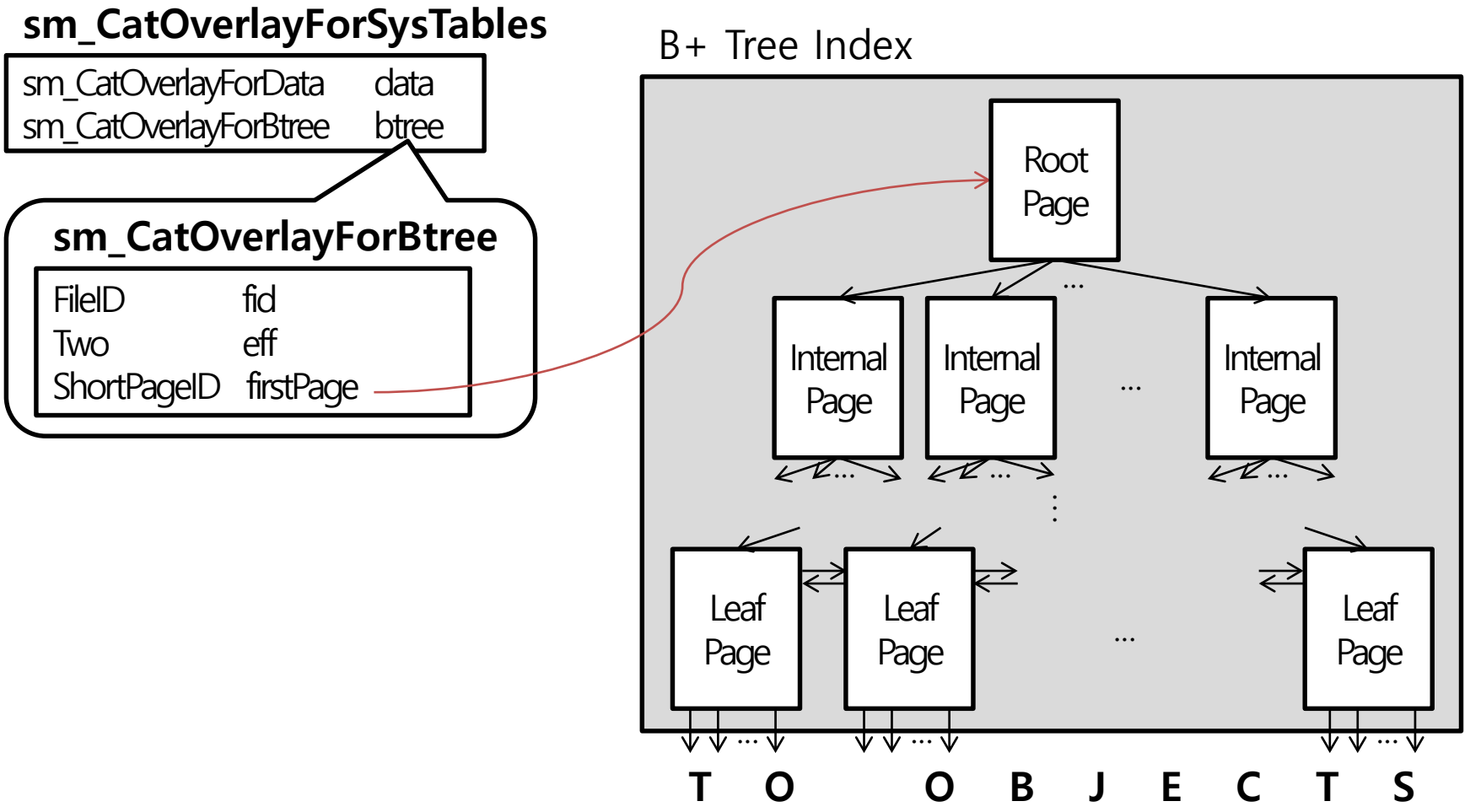
- Overview
  - A project for educational purposes where students implement a part of the coarse-granule locking version of the ODYSSEUS/COSMOS storage system
    - Prerequisites for the project: basic C programming skill
- Objective
  - To learn the functions of each module of a DBMS by implementing a part of the ODYSSEUS/COSMOS storage system
- Project types
  - EduBfM
    - We implement the operations of the buffer manager.
  - EduOM
    - We implement the operations of the object manager and the page-related structures.
  - EduBtM
    - We implement the operations of the B+ tree index manager.

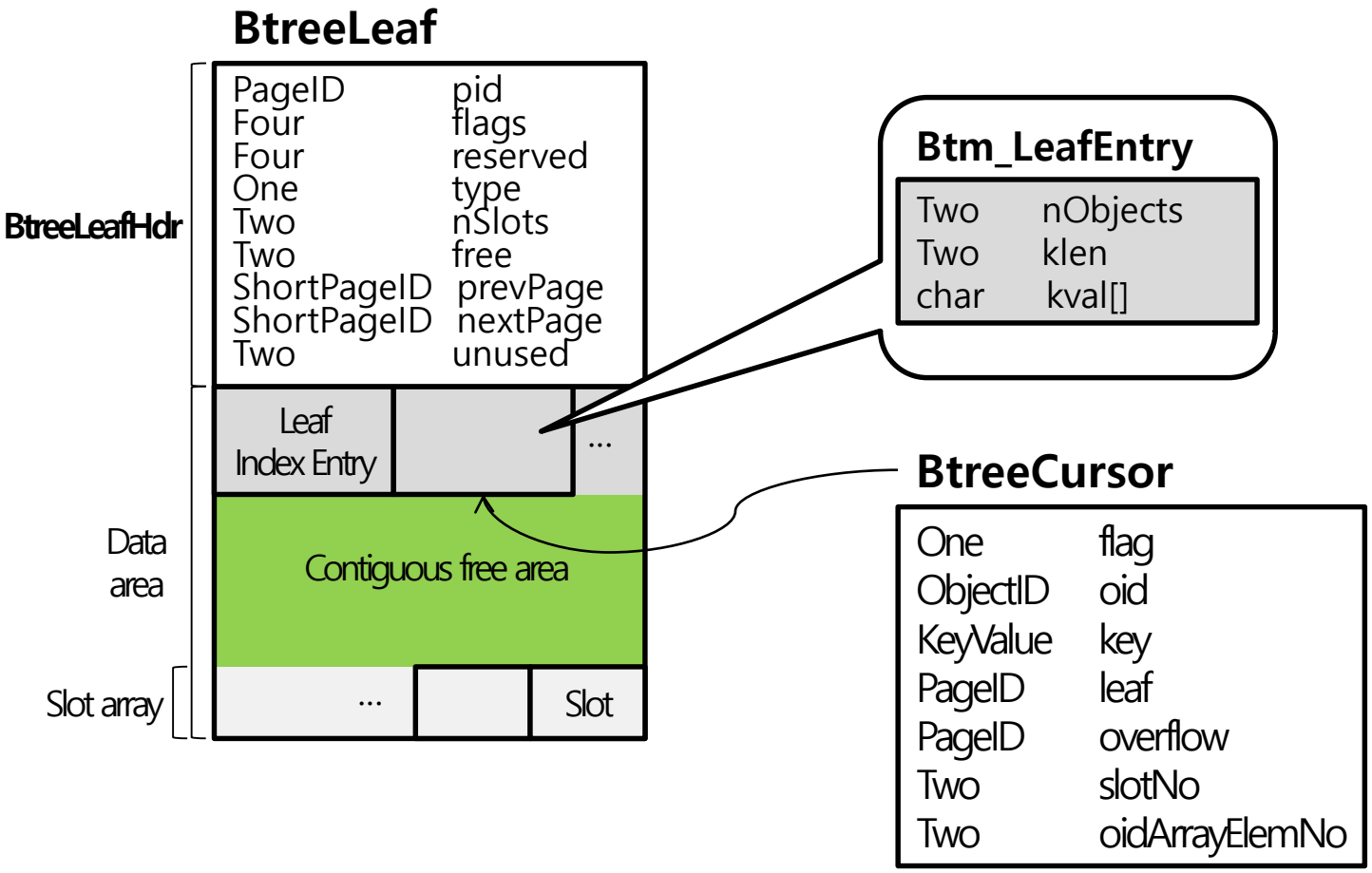
EduBtM Project

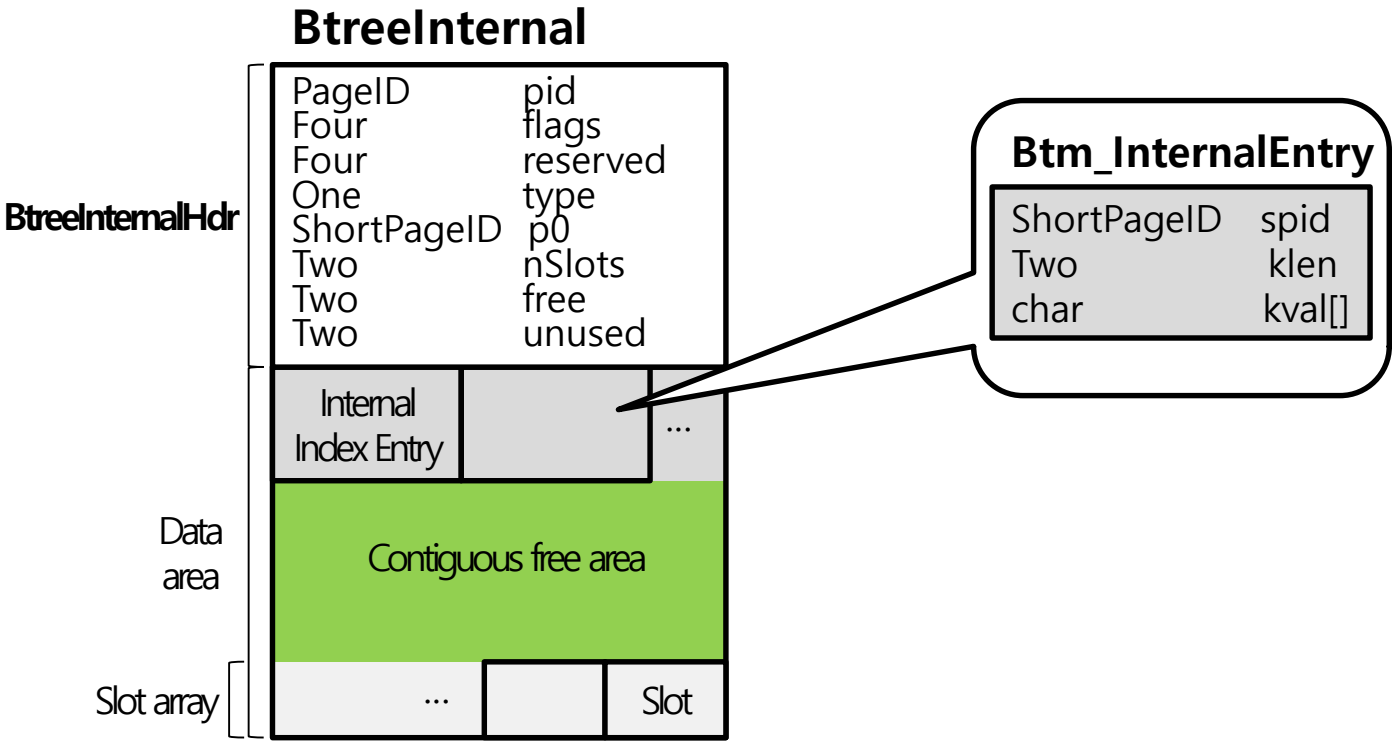
- Objective
  - We implement the operations of the B+ tree index and of the index page related structure.
  - In EduBtM, we handle only a very limited subset of original ODYSSEUS/COSMOS BtM functionality.



Data Structures







sm_CatOverlayForSysTables

- Overview
  - Data structure to store information about the data file and related index file.
    - Data file: A set of pages storing related objects
    - Index file: A set of pages of a B+ tree index created for the data file

- Components
  - data
    - Data structure to store information about the data file
      - Refer to the EduOM project manual
  - btree
    - Data structure to store information about an index file

sm_CatOverlayForBtree

- Overview
  - Data structure to store information about an index file

- Components
  - fid
    - ID of the index file
  - eff
    - Extent fill factor of an index file, which is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - firstPage
    - Page number of the first page of the index file
      - The page number of the root of a newly created B+ tree index

BtreeLeaf

- Overview
  - The page data structure representing a leaf node of a B+ tree index
- Components
  - hdr
    - Page header storing information about the page
  - data[]
    - Data area of a page storing leaf index entries
      - An <object key, object ID> pair and related information is stored as an index entry
      - Data area must always be filled more than 50%.
        - » Exception: when the page is the root

- slot[1]
  - Array of slots storing the offsets of the index entries stored in the data area of the page.
    - All the slots (*slot*[-1] ~ *slot*[-*n*]) except the first slot (*slot*[0]) share the memory space with the data area.
      - » Array index of the first slot = 0
      - » Array index of the next slot
        = Array index of the previous slot – 1
      - » Slot number = |Array index of the slot|
    - For efficient search, the index entries' offsets are stored in the slots as sorted in the order of the keys of the index entries.
      - » The first slot: the offset of the index entry with the smallest key in the page is stored.
      - » The last slot: the offset of the index entry with the largest key in the page is stored.
      - » We do not need to sort the index entries themselves in the data area.

BtreeLeafHdr

- Overview
  - Data structure to store information about a leaf page of a B+ tree index.

- Components
  - pid
    - ID of the page
  - flags
    - Set of bits indicating the page's type
      - If both the first and third bits are set (BTREE_PAGE_TYPE): indicates that the page is a B+ tree index page.
      - Other bits are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
  - reserved
    - Reserved variable to store additional information about the page
  - type
    - Set of bits indicating the type of a B+ tree index page
      - If the first bit is set (ROOT): indicates a root page.
      - If the second bit is set (INTERNAL): indicates an internal page.

    - If the third bit is set (LEAF): indicates a leaf page.
    - If both the second and fourth bits are set (FREEPAGE): indicates a page to be deallocated.
    - Other bits are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
- nSlots
  - Size of the slot array of the page
    (= The last slot number of the slots being used + 1)
    - The size of the slot array is changed dynamically as an index entry is inserted/deleted.
- free
  - Starting offset of the contiguous free area in the page's data area.
    - Contiguous free area: continuous free area after the last index entry in the data area. (※ When deleting the last index entry, the contiguous free area after the start offset of the entry deleted is the contiguous free area.)
- prevPage / nextPage
  - Page number of the next/previous leaf page
    - Used for maintaining the doubly linked list structure among leaf pages of the B+ tree index.
- unused
  - Total size of free areas except the contiguous free area in the page's data area (unit: bytes)

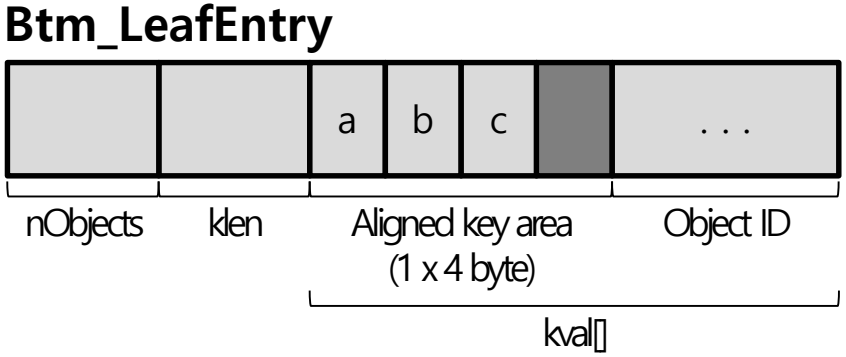
Btm_LeafEntry

- Overview
  - Data structure representing a leaf index entry stored in a leaf page of a B+ tree index

- Components
  - nObjects
    - The number of objects in the index entry with the same key
      - In EduBtM, we handle only one object corresponding to each key (i.e., we handle only a unique key).
  - klen
    - Length of the key stored in the index entry (unit: bytes)
      - The actual length of the key (but not the size of the aligned key area).

– kval[]

- Data area storing the <object's key, object ID> pairs
  - The area storing an object's key is aligned to be a multiple of 4 (the basic unit of memory allocation in a 32-bit operating system).
    - » Example) when a key 'abc', whose length is 3, is stored,



BtreeInternal

- Overview
  - The page data structure representing an internal node of a B+ tree index

- Components
  - hdr
    - Page header storing information about the page.
  - data[]
    - Data area of a page storing internal index entries.
      - An <object's key, child page's page number> pair and related information is stored as an index entry.
      - Data area must always be filled more than 50%.
        - » Exception: when the page is the root

– slot[1]
  - Array of slots storing the offsets of the index entries stored in the data area of the page.
    – All the slots (*slot*[-1] ~ *slot*[-*n*]) except the first slot (*slot*[0]) share the memory space with the data area.
      » Array index of the first slot = 0
      » Array index of the next slot
        = Array index of the previous slot – 1
      » Slot number = |Array index of the slot|
    – For efficient search, the index entries' offsets are stored in the slots as sorted in the order of the keys of the index entries.
      » The first slot: the offset of the index entry with the smallest key in the page is stored.
      » The last slot: the offset of the index entry with the largest key in the page is stored.
      » We do not need to sort the index entries themselves in the data area.

BtreeInternalHdr

- Overview
  - Data structure to store information about an internal page of a B+ tree index

- Components
  - pid
    - ID of the page
  - flags
    - Set of bits indicating the page's type
      - If both the first and third bits are set (BTREE_PAGE_TYPE): indicates that the page is a B+ tree index page.
      - Other bits are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
  - reserved
    - Reserved variable to store additional information about the page
  - type
    - Set of bits indicating the type of a B+ tree index page
      - If the first bit is set (ROOT): indicates a root page.
      - If the second bit is set (INTERNAL): indicates an internal page.

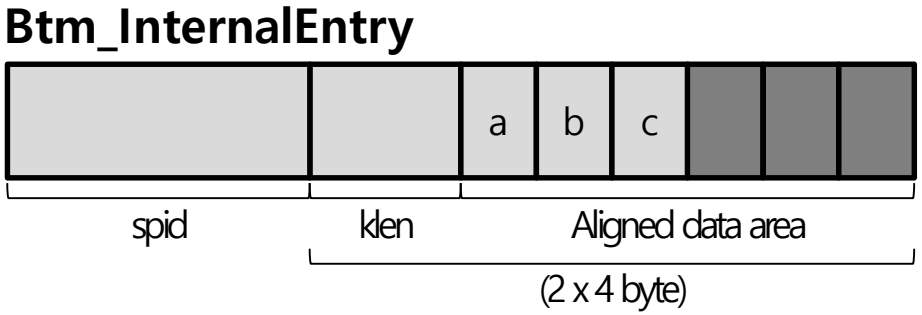
  - If the third bit is set (LEAF): indicates a leaf page.
  - If both the second and fourth bits are set (FREEPAGE): indicates a page to be deallocated.
  - Other bits are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
- p0
  - The page number of a child page storing index entries with keys smaller than the key of the first index entry (slot number = 0) of the page.
- nSlots
  - Size of the slot array of the page
    (= The last slot number of the slots being used + 1)
    - The size of the slot array is changed dynamically as an index entry is inserted/deleted.
- free
  - Starting offset of the contiguous free area in the page's data area
- unused
  - Total size of free areas except the contiguous free area in the page's data area (unit: bytes)

Btm_InternalEntry

- Overview
  - Data structure representing an internal index entry stored in an internal page of a B+ tree index
- Components
  - spid
    - The page number of a child page storing index entries with keys greater than or equal to the key of the current index entry and smaller than the key of the next index entry
  - klen
    - Length of the key stored in an index entry (unit: bytes)
      - The actual length of the key (but not the size of the aligned key area).

– kval[]

- Data area storing the discriminator key
  - The area storing the discriminator key + *klen* is aligned to be a multiple of 4 (the basic unit of memory allocation in a 32-bit operating system).
    - » Example) when a key "abc", whose length is 3, is stored,

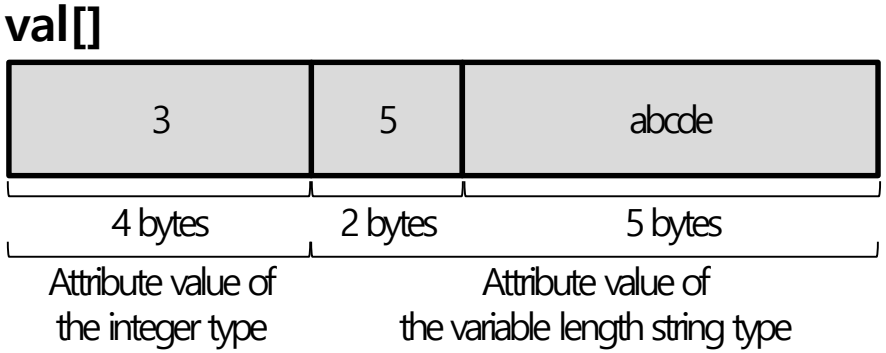


KeyValue

- Overview
  - Data structure to store a key and related information used in a B+ tree index
  - Possible to store multi-attribute keys and variable length keys.
    - Multi-attribute key: a key composed of two or more attributes.

- Components
  - len
    - Length of the key
  - val[]
    - Sequence of attribute values of the key
      - Information, which is needed to separate individual attribute values in a sequence, is stored in the data structure *KeyDesc*.

– In case of an attribute value of the variable length string type, the attribute value is stored together with its length.
  » Example) key value composed of 3 (an attribute value of the integer type) and "abcde" (an attribute value of the variable length string type)



KeyDesc

- Overview
  - Data structure to store information for separating individual attribute values in a sequence of attribute values of a key

- Components
  - flag
    - Set of bits indicating the type of a key
      - If the first bit is set (KEYFLAG_UNIQUE): indicates a unique key.
      - Other bits are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
  - nparts
    - Number of attribute values in a key

- kpart[]
  - Array data structure storing information about each attribute of a key
    - type
      - » Type of the attribute
        - In case of the integer type, *type* := SM_INT
        - In case of the variable length string type, *type* := SM_VARSTRING
        - Other types are not used in EduBtM. (You may ignore them when implementing the EduBtM function.)
    - offset
      - » Offset of the attribute value residing in array *val*[] of *KeyValue*. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
    - length
      - » Length of the attribute value
      - » This is not used in the case of an attribute value of the variable length string type.

BtreeCursor

- Overview
  - Data structure of the cursor pointing to a leaf index entry of the B+ tree index
  - Used for storing the information about the current leaf index entry being searched and the next leaf index entry to be searched when sequentially searching for objects satisfying the search condition.

- Components
  - flag
    - Variable indicating the cursor's status
      - CURSOR_ON: indicates that the cursor points to a leaf index entry satisfying the search condition.
      - CURSOR_EOS: indicates that there is no leaf index entry satisfying the search condition (meaning the end of search).
  - oid
    - Object ID (OID) stored in the leaf index entry pointed to by the cursor

- key
  - Key value of the object stored in the leaf index entry pointed to by the cursor
- leaf
  - Page ID of the leaf page storing the leaf index entry pointed to by the cursor
- overflow
  - Page ID of the page storing the IDs of objects (OIDs) with the same key value when duplicated keys are allowed. This is not used in EduBtM allowing only unique keys. (You may ignore it when implementing the EduBtM function.)
- slotNo
  - The slot number of the leaf index entry pointed to by the cursor.
- oidArrayElmNo
  - Index of the array storing the IDs of objects (OIDs) with the same key value when using duplicated keys. This is not used in EduBtM allowing only unique keys. (You may ignore it when implementing the EduBtM function.)

Related Operations

- Create / Delete a B+ tree index.
- Insert a new object into the B+ tree index.
  - Consider the split operation when insertion causes overflow.
- Delete an object from the B+ tree index
  - Implementation of the delete operation is optional (not mandatory).
    - An extra credit will be given.
  - Handle the merge/redistribute operation when deletion causes underflow by using the functions given.
- Search for an object in the B+ tree index.

API Functions to Implement

- EduBtM_CreateIndex()
- EduBtM_DropIndex()
- EduBtM_InsertObject()
- EduBtM_DeleteObject()
- EduBtM_Fetch()
- EduBtM_FetchNext()

(※ API functions mean they are part of the ODYSSEUS/COSMOS API shown in p.4)
(※ API: Application Programming Interface)

EduBtM_CreateIndex()

- File: EduBtM_CreateIndex.c

- Description
  - Create a new B+ tree index in an index file, and return the page ID of the root page of index created.
    - Call btm_AllocPage() to allocate the first page of the index file.
      - The page number of the first page is stored in the variable *firstPage* of *sm_CatOverlayForBtree*.
    - Initialize the allocated page as the root page.
    - Return the page ID of the root page initialized.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file in which a B+ tree index is created and about the data file to be indexed
  - PageID *rootPid
    (OUT) Page ID of the root page of the B+ tree index created

- Return value
  - Four error code

- Related functions
  edubtm_InitLeaf(), btm_AllocPage(), BfM_GetTrain(), BfM_FreeTrain()

EduBtM_DropIndex()

- File: EduBtM_DropIndex.c

- Description
  - Delete a B+ tree index from an index file.
    - Deallocate a root page and every child page of the B+ tree index.

- Parameters
  - PhysicalFileID *pFid
    (IN) File ID of the index file containing the B+ tree index to be deleted (= page ID of the first page of the index file)
  - PageID *rootPid
    (IN) Page ID of the root page of the B+ tree index to be deleted
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list

- Return value
  - Four error code

- Related functions
  edubtm_FreePages()

EduBtM_InsertObject()

- File: EduBtM_InsertObject.c

- Description
  - Insert a new object into a B+ tree index.
    - Call edubtm_Insert() to insert the <object's key, object ID> pair for the new object into the B+ tree index.
    - If it is necessary to create a new root page because the root page has been split, call edubtm_root_insert() to handle it.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index to which to insert the object
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the object to be inserted
  - ObjectID *oid
    (IN) OID of the object to be inserted
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)

- Return value
  - Four error code

- Related functions
  edubtm_Insert(), edubtm_root_insert(), BfM_GetTrain(), BfM_FreeTrain()

EduBtM_DeleteObject()

- File: EduBtM_DeleteObject.c

- Description
  - Delete an object from a B+ tree index
    - Call edubtm_Delete() to delete the <object's key, object ID> pair for the object to be deleted from the B+ tree index.
    - If underflow has occurred in the root page, call btm_root_delete() to handle it.
      - btm_root_delete() checks whether the root page is empty; if so, delete the root.
    - If the root page has been split, call edubtm_root_insert() to handle it.
      - The root page may split in the process of redistribution since an index entry in the root may have been replaced with another whose length is longer than itself.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index from which to delete an object
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the object to be deleted
  - ObjectID *oid
    (IN) OID of the object to be deleted
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list

- Return value
  - Four error code

- Related functions
  edubtm_root_insert(), edubtm_Delete(), btm_root_delete(), BfM_GetTrain(), BfM_FreeTrain()

EduBtM_Fetch()

- File: EduBtM_Fetch.c

- Description
  - Search for the first object satisfying the search condition in the B+ tree index, and return the cursor pointing to the object found.
    - If *startCompOp* given as a parameter is SM_BOF,
      - Search for the first object (the leaf index entry with the smallest key value) in the B+ tree index.
    - If *startCompOp* given as a parameter is SM_EOF,
      - Search for the last object (the leaf index entry with the largest key value) in the B+ tree index.
    - Other cases,
      - Call edubtm_Fetch() to search for a leaf index entry containing the first <object's key, object ID> pair satisfying the search condition in the B+ tree index.
    - Return the cursor pointing to the leaf index entry found.

- Parameters
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *startKval
    (IN) Start key value for search
  - Four startCompOp
    (IN) Comparison operator to apply to the start key value
  - KeyValue *stopKval
    (IN) Stop key value for search
  - Four stopCompOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *cursor
    (OUT) Cursor pointing to the leaf index entry corresponding to the first object satisfying the search condition

- Return value
  - Four error code

- Related functions
  edubtm_Fetch(), edubtm_FirstObject(), edubtm_LastObject()

EduBtM_FetchNext()

- File: EduBtM_FetchNext.c

- Description
  - Fetches the object next to the current object satisfying the search condition (considering only the stop key value but not start key value) in the B+ tree index, and return the cursor pointing to the object found.
    - Call edubtm_FetchNext() to search for the leaf index entry next to the current leaf index entry satisfying the search condition in the B+ tree index.
    - Return the cursor pointing to the leaf index entry found.

- Parameters
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Stop key value for searching
  - Four compOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *current
    (IN) Cursor pointing to the leaf index entry corresponding to the current object satisfying the search condition
  - BtreeCursor *next
    (OUT) Cursor pointing to the leaf index entry corresponding to the next object satisfying the search condition

- Return value
  - Four error code

- Related functions
  edubtm_FetchNext(), edubtm_KeyCompare(), BfM_GetTrain(), BfM_FreeTrain()

Internal Functions to Implement

- edubtm_InitLeaf()
- edubtm_InitInternal()
- edubtm_FreePages()
- edubtm_Insert()
- edubtm_InsertLeaf()
- edubtm_InsertInternal()
- edubtm_SplitLeaf()
- edubtm_SplitInternal()
- edubtm_root_insert()
- edubtm_Delete()
- edubtm_DeleteLeaf()
- edubtm_CompactLeafPage()
- edubtm_CompactInternalPage()
- edubtm_Fetch()
- edubtm_FetchNext()
- edubtm_FirstObject()
- edubtm_LastObject()
- edubtm_BinarySearchLeaf()
- edubtm_BinarySearchInternal()
- edubtm_KeyCompare()

edubtm_InitLeaf()

- File: edubtm_InitPage.c

- Description
  - Initialize a page as a leaf page of the B+ tree index.
    - Initialize the page header as a leaf page.
      - *pid* := Page ID given as a parameter
      - *flags*
        - » Set a bit indicating that the page is a B+ tree index page
      - *type*
        - » Set a bit indicating that the page is a leaf page
        - » If *root* given as a parameter is TRUE, set a bit indicating that the page is a root page.
      - *nSlots* := 0
      - *free* := 0
      - *prevPage* := NIL
      - *nextPage* := NIL
      - *unused* := 0

- Parameters
  - PageID *leaf
    (IN) Page ID of the page to be initialized
  - Boolean root
    (IN) Flag indicating that the page to be initialized is a root page
  - Boolean isTmp
    (IN) Flag indicating that the page to be initialized is a temporary page. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)

- Return value
  - Four error code

- Related functions
  BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_InitInternal()

- File: edubtm_InitPage.c

- Description
  - Initialize a page as an internal page of the B+ tree index
    - Initialize the page header as an internal page
      - *pid* := Page ID given as a parameter
      - *flags*
        - » Set a bit indicating that the page is a B+ tree index page
      - *type*
        - » Set a bit indicating that the page is an internal page
        - » If *root* given as a parameter is TRUE, set a bit indicating that the page is a root page.
      - *p0* := NIL
      - *nSlots* := 0
      - *free* := 0
      - *unused* := 0

- Parameters
  - PageID *internal
    (IN) Page ID of the page to be initialized
  - Boolean root
    (IN) Flag indicating that the page to be initialized is a root page
  - Boolean isTmp
    (IN) Flag indicating that the page to be initialized is a temporary page. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)

- Return value
  - Four error code

- Related functions
  BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_FreePages()

- File: edubtm_FreePages.c

- Description
  - Deallocate a B+ tree index page.
    - Call edubtm_FreePages() recursively for every child page of the page given as a parameter to deallocate the corresponding pages.
    - Deallocate a page given as a parameter.
      - In the page header's *type*, set a bit indicating that the page is to be deallocated, and unset other bits.
      - Deallocate the corresponding page.
        - » Allocate a new dealloc list element from the *dlPool* given as a parameter
          - Dealloc list: A linked list of pages to be deallocated
        - » Store the information about the page to be deallocated into the element allocated.
        - » Insert an element containing the information about the page to be deallocated into the first element of the dealloc list.

- Parameters
  - PhysicalFileID *pFid
    (IN) File ID of the index file containing the page to be deallocated (= page ID of the first page of the index file)
  - PageID *curPid
    (IN) Page ID of the page to be deallocated
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list

- Return value
  - Four error code

- Related functions
  BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty(), Util_getElementFromPool()

edubtm_Insert()

- File: edubtm_Insert.c

- Description
  - Insert an <object's key, object ID> pair for a new object into the B+ tree whose root page is given as a parameter, and if split occurs in the root page, return the internal index entry pointing to the new page created by the split.
    - If the root page given as a parameter is an internal page,
      - Determine the next child page to visit to find the leaf page to insert the <object's key, object ID> pair.
      - Call edubtm_Insert() recursively to insert the <object's key, object ID> pair into the B+ subtree having the child page as the root.

  - If split occurs in the child page determined, insert the internal index entry pointing to the new page created by the split into the root page given as a parameter.
    - » Determine the position (slot number) to insert the index entry.
      - Offsets of the index entries must be stored in the slot array in a sorted order of the keys of the index entries.
    - » Call edubtm_InsertInternal() to insert the index entry with the slot number determined.
  - If split occurs in the root page given as a parameter, return the internal index entry pointing to the new page created by the split.
- If the root page given as a parameter is a leaf page,
  - Call edubtm_InsertLeaf() to insert the <object's key, object ID> pair into the page.
  - If split occurs, return the internal index entry pointing to the new page created by the split.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index. (Since edubtm_Insert() is called recursively, the type of this page may not be the root page.)
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the object to be inserted
  - ObjectID *oid
    (IN) OID of the object to be inserted
  - Boolean *f
    (OUT) Flag indicating that the root page has been merged. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - Boolean *h
    (OUT) Flag indicating that the root page has been split
  - InternalItem *item
    (OUT) Internal index entry pointing to the new page created by the split of the root page
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)

- Return value
  - Four error code

- Related functions
  edubtm_InsertLeaf(), edubtm_InsertInternal(), edubtm_BinarySearchInternal(), BfM_GetTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_InsertLeaf()

- File: edubtm_Insert.c
- Description
  - Insert a new index entry into a leaf page, and if split occurs, return the internal index entry pointing to the new leaf page created by the split.
    - Determine the position (slot number) to insert the index entry.
      - Offsets of the index entries must be stored in the slot array in a sorted order of the keys of the index entries.
      - If an index entry whose key value is the same as that of the index entry to be inserted exists, return error eDUPLICATEDKEY_BTM.
    - Calculate the size of free area required for inserting the new index entry.
      - Size of the new index entry considering the key area aligned + size of the slot
    - If there is available free area in the page,
      - Compact the page if necessary.
      - Insert the new index entry with the slot number determined.
        - » Copy the new index entry into the contiguous free area of the page.
        - » Rearrange the slot array to use the slot with the slot number determined.
        - » Store the offset of the new index entry into the slot with the slot number determined.
        - » Update the page's header.
    - If there is no available free area in the page (page overflow),
      - Call edubtm_SplitLeaf() to split the page.
      - Return the internal index entry pointing to the new leaf page created by the split.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *pid
    (IN) Page ID of the leaf page to insert the index entry into
  - BtreeLeaf *page
    (INOUT) Pointer to the buffer element containing the leaf page to insert the index entry into
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the index entry to be inserted
  - ObjectID *oid
    (IN) Object ID (OID) stored in the index entry to be inserted
  - Boolean *f
    (OUT) Flag indicating that the page has been merged. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - Boolean *h
    (OUT) Flag indicating that the page has been split
  - InternalItem *item
    (OUT) Internal index entry pointing to the new leaf page created by the split

- Return value
  - Four error code

- Related functions
  edubtm_SplitLeaf(), edubtm_CompactLeafPage(), edubtm_BinarySearchLeaf()

edubtm_InsertInternal()

- File: edubtm_Insert.c
- Description
  - Insert a new index entry into an internal page, and if split occurs, return the internal index entry pointing to the new internal page created by the split.
    - Calculate the size of free area required for inserting the new index entry.
      - Size of the new index entry considering the key area aligned + size of the slot
    - If there is available free area in the page,
      - Compact the page if necessary.
      - Insert the new index entry with the slot number next to the slot number given as a parameter.
        - » Copy the new index entry into the contiguous free area of the page.
        - » Rearrange the slot array to use the slot with the slot number determined.
        - » Store the offset of the new index entry into the slot with the slot number determined.
        - » Update the page's header.
    - If there is no available free area in the page (page overflow),
      - Call edubtm_SplitInternal() to split the page.
      - Return the internal index entry pointing to the new internal page created by the split.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - BtreeInternal *page
    (INOUT) Pointer to the buffer element containing the internal page to insert the index entry into
  - InternalItem *item
    (IN) Index entry to be inserted (not aligned)
  - Two high
    (IN) Slot number of the index entry with the largest key value among the index entries with a key value smaller than that of the index entry to be inserted
  - Boolean *h
    (OUT) Flag indicating that the page has been split
  - InternalItem *ritem
    (OUT) Internal index entry pointing to the new internal page created by the split

- Return value
  - Four error code

- Related functions
  edubtm_SplitInternal(), edubtm_CompactInternalPage()

edubtm_SplitLeaf()

- File: edubtm_Split.c

- Description
  - Split a leaf page, in which overflow has occurred, to insert the index entry given as a parameter, and return the internal index entry pointing to the new leaf page created by the split.
    - Allocate a new page.
    - Initialize the allocated page as a leaf page.

- After sorting the existing index entries and the given index entry in the order of the keys, divide them into the overflowed page and the allocated page to store them.
  - First, store as many index entries as to occupy 50% of the total space for all the index entries.
  - Store the remaining index entries in the allocated page.
  - Update each page's header.
- Add the allocated page into the doubly linked list of the leaf pages.
  - Add the allocated page as the next page of the overflowed page.
- Create an internal index entry pointing to the allocated page.
  - Discriminator key value := key value of the first index entry (slot number = 0) of the allocated page
    - » In a B+ tree index, a key value (discriminator key value) of an internal index entry is duplicated from the key value of a leaf index entry.
  - Page number of the child page := page number of the allocated page
- Return the index entry created.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the page that overflowed
  - BtreeLeaf *fpage
    (INOUT) Pointer to the buffer element containing the page that overflowed
  - Two high
    (IN) Slot number of the index entry with the largest key value among the index entries with a key value smaller than that of the index entry to be inserted
  - LeafItem *item
    (IN) Index entry to be inserted (not aligned)
  - InternalItem *ritem
    (OUT) Internal index entry pointing to the new leaf page created by the split

- Return value
  - Four error code

- Related functions
  edubtm_InitLeaf(), edubtm_CompactLeafPage(), btm_AllocPage(), BfM_GetTrain(), BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_SplitInternal()

- File: edubtm_Split.c

- Description
  - Split an internal page, in which overflow has occurred, to insert the index entry given as a parameter, and return the internal index entry pointing to the new internal page created by the split.
    - Allocate a new page.
    - Initialize the allocated page as an internal page.
    - After sorting the existing index entries and the given index entry in the order of the keys, divide them into the overflowed page and the allocated page to store them.
      - First, store as many index entries as to occupy 50% of the total space for all the index entries.
      - Store the page number of the child page pointed to by the first index entry (1st entry) among those that have not been stored yet in the variable *p0* of the allocated page's header.
      - Set 1st entry as an internal index entry pointing to the allocated page, and return it.
        - » Page number of the child page := page number of the allocated page
      - Store the remaining index entries in the allocated page.
      - Update each page's header.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - BtreeLeaf *fpage
    (INOUT) Pointer to the buffer element containing the page that overflowed
  - Two high
    (IN) Slot number of the index entry with the largest key value among the index entries with a key value smaller than that of the index entry to be inserted
  - LeafItem *item
    (IN) Index entry to be inserted (not aligned)
  - InternalItem *ritem
    (OUT) Internal index entry pointing to the new internal page created by the split

- Return value
  - Four error code

- Related functions

  edubtm_InitInternal(), edubtm_CompactInternalPage(), btm_AllocPage(), BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_root_insert()

- File: edubtm_root.c
- Description
  - Create a new root page for the B+ tree index whose root page has been split.
    - Allocate a new page.
    - Copy the old root page into the page allocated.
    - Initialize the old root page as the new root page.
      - To keep the page ID of the root page of the B+ tree index consistently.
    - Make the page allocated and the page created by the split of the root page to be children pages of the new root page.
      - Insert the internal index entry pointing to the page created by the split into the new root page.
      - Store the page number of the allocated page in the variable *p0* of the new root page's header.
      - If the children pages of the new root page are leaf pages, set the doubly linked list between two children pages.
        - » Set the page created by the split to be the next page of the page allocated.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the root page split
  - InternalItem *ritem
    (IN) Internal index entry pointing to the new page created by the split of the root page

- Return value
  - Four error code

- Related functions
  edubtm_InitInternal(), btm_AllocPage(), BfM_GetTrain(), BfM_GetNewTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_Delete()

- File: edubtm_Delete.c

- Description
  - Delete an <object's key, object ID> pair from the B+ tree whose root page is given as a parameter.
    - If the root page given as a parameter is an internal page,
      - Determine the next child page to visit to find the leaf page containing the <object's key, object ID> pair to be deleted.
      - Call edubtm_Delete() recursively to delete the <object's key, object ID> pair from the B+ subtree having the child page as the root.
      - If underflow has occurred in the child page determined, call btm_Underflow() to handle it.
        - » If overflow occurred in the parent page (the root page given as a parameter) of the child page that underflowed, call edubtm_InsertInternal() to insert the internal index entry that was not inserted by the overflow into the parent page
          - Since the parent page will be split as a result of calling edubtm_InsertInternal(), set the out parameter *h* to TRUE and return the internal index entry pointing to the new page created by the split
        - » Since contents of the root page will be changed as a result of calling btm_Underflow(), you should set the DIRTY bit of the root page to 1 after calling btm_Underflow().
    - If the root page given as a parameter is a leaf page,
      - Call edubtm_DeleteLeaf() to delete the <object's key, object ID> pair from the page.
      - If underflow has occurred in the page (size of the free area of the data area of the page > (size of the total data area of the page / 2)), set the out parameter *f* to TRUE.

- Parameters
  - ObjectID *catObjForFile
    (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the object to be deleted
  - ObjectID *oid
    (IN) OID of the object to be deleted
  - Boolean *f
    (OUT) Flag indicating that underflow has occurred in the root page
  - Boolean *h
    (OUT) Flag indicating that the root page has been split
  - InternalItem *item
    (OUT) Internal index entry pointing to the new page created by the root page split
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list

- Return value
  - Four error code

- Related functions
  edubtm_InsertInternal(), edubtm_DeleteLeaf(), edubtm_BinarySearchInternal(), btm_Underflow(), BfM_GetTrain(), BfM_FreeTrain(), BfM_SetDirty()

edubtm_DeleteLeaf()

- File: edubtm_Delete.c

- Description
  - Delete an <object's key, object ID> pair from the leaf page.
    - Delete the slot containing the offset of the index entry containing the <object's key, object ID> pair to be deleted.
      - Compact the slot array so that there is no empty slot deleted in the middle of the slot array.
    - Update the leaf page's header.
    - If underflow has occurred in the leaf page (size of the free area of the data area of the page > (size of the total data area of the page / 2)), set the out parameter *f* to TRUE.

- Parameters
  - PhysicalFileID *pFid
    (IN) File ID of the index file containing the B+ tree index (= page ID of the first page of the index file)
  - PageID *pid
    (IN) Page ID of the leaf page from which to delete an object
  - BtreeLeaf *apage
    (INOUT) Pointer to the buffer element containing the leaf page from which to delete an object
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value of the index entry to be deleted
  - ObjectID *oid
    (IN) Object ID (OID) stored in the index entry to be deleted
  - Boolean *f
    (OUT) Flag indicating that underflow has occurred in the leaf page
  - Boolean *h
    (OUT) Flag indicating that the leaf page has been split. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - InternalItem *item
    (OUT) Internal index entry pointing to the new page created by the split of the leaf page. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - Pool *dlPool
    (INOUT) Pool from which to allocate a new dealloc list element. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)
  - DeallocListElem *dlHead
    (INOUT) Header pointing to the first element of the dealloc list. This is not used in EduBtM. (You may ignore it when implementing the EduBtM function.)

- Return value
  - Four error code

- Related functions
  edubtm_BinarySearchLeaf(), btm_ObjectIdComp(), BfM_SetDirty()

edubtm_CompactLeafPage()

- File: edubtm_Compact.c

- Description
  - Adjust offsets of index entries so that every free area in the leaf page's data area forms an uninterrupted contiguous free area.
    - If *slotNo* given as a parameter is not NIL,
      - Contiguously store all the index entries in the page except for the index entry corresponding to *slotNo* from the very front of the data area.
        - » Order of storing index entries: order of the corresponding slot numbers
      - Store the index entry corresponding to *slotNo* as the last one in the data area.
    - If *slotNo* given as a parameter is NIL,
      - Store every index entry in the page contiguously from the very front of the data area.
        - » Order of storing index entries: order of the corresponding slot numbers
    - Update the page header.

- Parameters
  - BtreeLeaf *apage

    (INOUT) Pointer to the buffer element containing the leaf page to be compacted
  - Two slotNo

    (IN) Slot number of the index entry that is to be stored as the last one in the data area of the page
- Return value
  - Four error code
- No related functions

edubtm_CompactInternalPage()

- File: edubtm_Compact.c

- Description
  - Adjust offsets of index entries so that every free area in the internal page's data area forms an uninterrupted contiguous free area.
    - If *slotNo* given as a parameter is not NIL,
      - Contiguously store all the index entries in the page except for the index entry corresponding to *slotNo* from the very front of the data area.
        - » Order of storing index entries: order of the corresponding slot numbers
      - Store the index entry corresponding to *slotNo* as the last one in the data area.
    - If *slotNo* given as a parameter is NIL,
      - Store every index entry in the page contiguously from the very front of the data area.
        - » Order of storing index entries: order of the corresponding slot numbers
    - Update the page header.

- Parameters
  - BtreeInternal *apage

    (INOUT) Pointer to the buffer element containing the internal page to be compacted
  - Two slotNo

    (IN) Slot number of the index entry that is to be stored as the last one in the data area of the page
- Return value
  - Four error code
- No related functions

edubtm_Fetch()

- File: EduBtM_Fetch.c

- Description
  - Search for a leaf index entry containing the first <object's key, object ID> pair satisfying the search condition in the B+ tree index whose root page is given as a parameter; return the cursor pointing to the leaf index entry found.
    - If the root page given as a parameter is an internal page,
      - Determine the next child page to visit to find a leaf page containing the first <object's key, object ID> pair satisfying the search condition.
      - Call edubtm_Fetch() recursively to search for a leaf index entry containing the first <object's key, object ID> pair satisfying the search condition in the B+ subtree having the child page as the root.
      - Return the cursor pointing to the leaf index entry found.

- If the root page given as a parameter is a leaf page,
  - Search for an index entry containing the first <object's key, object ID> pair satisfying the search condition.
    - » The search condition is specified by the start/end key values and the comparison operator given as parameters.
      - SM_EQ: satisfied when the key value of the object is equal to the start/end key value
      - SM_LT: satisfied when the key value of the object is smaller than the start/end key value
      - SM_LE: satisfied when the key value of the object is equal to or smaller than the start/end key value
      - SM_GT: satisfied when the key value of the object is greater than the start/end key value
      - SM_GE: satisfied when the key value of the object is equal to or greater than the start/end key value
  - Return the cursor pointing to the index entry found.

- Parameters
  - PageID *root
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *startKval
    (IN) Start key value
  - Four startCompOp
    (IN) Comparison operator to apply to the start key value
  - KeyValue *stopKval
    (IN) Stop key value
  - Four stopCompOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *cursor
    (OUT) Cursor pointing to the leaf index entry containing the first <object's key, object ID> pair satisfying the search condition

- Return value
  - Four error code

- Related functions
  edubtm_BinarySearchLeaf(), edubtm_BinarySearchInternal(), edubtm_KeyCompare(), BfM_GetTrain(), BfM_FreeTrain()

edubtm_FetchNext()

- File: EduBtM_FetchNext.c
- Description
  - Search for the leaf index entry satisfying the search condition next to the current leaf index entry in the B+ tree index; return the cursor pointing to the leaf index entry found.
    - Search for the next leaf index entry satisfying the search condition.
      - The search condition is specified by the end key values and the comparison operator given as parameters.
        - » SM_EQ: satisfied when the key value of the object is equal to the end key value
        - » SM_LT: satisfied when the key value of the object is smaller than the end key value
        - » SM_LE: satisfied when the key value of the object is equal to or smaller than the end key value
        - » SM_GT: satisfied when the key value of the object is greater than the end key value
        - » SM_GE: satisfied when the key value of the object is equal to or greater than the end key value
    - Return the cursor pointing to the leaf index entry found.

- Parameters
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Stop key value
  - Four compOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *current
    (IN) Cursor pointing to the current leaf index entry satisfying the search condition
  - BtreeCursor *next
    (OUT) Cursor pointing to the next leaf index entry satisfying the search condition

- Return value
  - Four error code

- Related functions
  edubtm_KeyCompare(), BfM_GetTrain(), BfM_FreeTrain()

edubtm_FirstObject()

- File: edubtm_FirstObject.c

- Description
  - Search for the first object (the leaf index entry with the smallest key value) in the B+ tree index.
    - Return the cursor pointing to the first leaf index entry of the first leaf page in the B+ tree index.

- Parameters
  - PageID *root
    (IN) Page ID of the root page in the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *stopKval
    (IN) Stop key value
  - Four stopCompOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *cursor
    (OUT) Cursor pointing to the leaf index entry corresponding to the first object in the B+ tree index
- Return value
  - Four error code
- Related functions
  edubtm_KeyCompare(), BfM_GetTrain(), BfM_FreeTrain()

edubtm_LastObject()

- File: edubtm_LastObject.c

- Description
  - Search for the last object (the leaf index entry with the greatest key) in the B+ tree index.
    - Return the cursor pointing to the last index entry (slot number = *nSlots* - 1) of the last leaf page in the B+ tree index.

- Parameters
  - PageID *root
    (IN) Page ID of the root page in the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *stopKval
    (IN) Stop key value
  - Four stopCompOp
    (IN) Comparison operator to apply to the stop key value
  - BtreeCursor *cursor
    (OUT) Cursor pointing to the leaf index entry corresponding to the last object in the B+ tree index
- Return value
  - Four error code
- Related functions
  edubtm_KeyCompare(), BfM_GetTrain(), BfM_FreeTrain()

edubtm_BinarySearchLeaf()

- File: edubtm_BinarySearch.c

- Description
  - Search for an index entry in the leaf page with a key value equal to or smaller than that given by a parameter, and return the position (slot number) of the index entry found.
    - If there is an index entry with the same key value as that given by the parameter,
      - Return the index entry's slot number and TRUE.
    - If there is no index entry with the same key value as that given by the parameter,
      - Return the slot number of the index entry with the largest key value among the index entries with a key value smaller than that given by the parameter and FALSE.
    - If the key value given by the parameter is smaller than that of any index entry in the page,
      - Set the OUT parameter *idx* to -1, and return FALSE.

- Parameters
  - BtreeLeaf *lpage
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value used for searching
  - Two *idx
    (OUT) Slot number of the index entry found

- Return value
  - Four TRUE or FALSE

- Related function
  edubtm_KeyCompare()

edubtm_BinarySearchInternal()

- File: edubtm_BinarySearch.c

- Description
  - Search for an index entry in the internal page with a key value equal to or smaller than that given by a parameter, and return the position (slot number) of the index entry found.
    - If there is an index entry with the same key value as that given by the parameter,
      - Return the index entry's slot number and TRUE.
    - If there is no index entry with the same key value as that given by the parameter,
      - Return the slot number of the index entry with the largest key value among the index entries with a key value smaller than that given by the parameter and FALSE.

- Parameters
  - BtreeLeaf *lpage
    (IN) Page ID of the root page of the B+ tree index
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *kval
    (IN) Key value used for searching
  - Two *idx
    (OUT) Slot number of the index entry found

- Return value
  - Four TRUE or FALSE

- Related function
  edubtm_KeyCompare()

edubtm_KeyCompare()

- File: edubtm_Compare.c

- Description
  - Compare two key values given by parameters, and return the comparison result.
    - If the two key values are the same, return EQUAL.
    - If the first key value is greater, return GREATER.
    - If the first key value is smaller, return LESS.

- Parameters
  - KeyDesc *kdesc
    (IN) Information for separating individual attribute values of the key
  - KeyValue *key1
    (IN) First key value to be compared
  - KeyValue *key2
    (IN) Second key value to be compared

- Return value
  - Four EQUAL, GREATER, or LESS

- No related functions

Given API Functions

- BfM_GetTrain()
  - Fix a page/train in the buffer element, and return the pointer to the buffer element.
    - Every transaction should fix the page/train to the buffer before accessing it.
    - To fix the page/train that is newly allocated on the disk to the buffer, call BfM_GetNewTrain() rather than BfM_GetTrain() for performance reasons.
      - BfM_GetNewTrain() fixes the empty page/train to a buffer without accessing the disk since it is not necessary to access the disk to read the contents of that page/train, which is empty.
  - Parameters
    - TrainID *trainId
      (IN) Page ID of the page or the first page of the train to be fixed
    - char **retBuf
      (OUT) Pointer to the buffer element containing the page fixed
    - Four type
      (IN) Type of the buffer
  - Return value
    - Four error code

– Example

```
Four edubtm_Fetch(
    PageID              *root,     /* IN: ID of the current root page of the subtree */
    ...)
{
    BtreePage *apage;     /* pointer to a buffer */
    ...
    /* Fix the page to the buffer */
    e = BfM_GetTrain(root, (char **)&apage, PAGE_BUF);
    if (e < 0)  ERR(e);
    ...
}
```

- BfM_GetNewTrain()
  - Fix a page/train that has been newly allocated on the disk to the buffer element, and return the pointer to the buffer element.
  - Parameters
    - TrainID *trainId
      (IN) Page ID of the page or the first page of the train to be fixed
    - char **retBuf
      (OUT) Pointer to the buffer element containing the page fixed
    - Four type
      (IN) Type of the buffer
  - Return value
    - Four error code

– Example

```
Four edubtm_InitInternal(
    PageID  *internal,    /* IN: ID of the page to be initialized */
    ...)
{
    BtreeInternal *page;    /* pointer to a buffer */
    ...
    /* Fix the page that has been newly allocated on the disk to the buffer */
    e = BfM_GetNewTrain(internal, (char **)&page, PAGE_BUF);
    if (e < 0)  ERR(e);
    ...
}
```

- BfM_FreeTrain()
  - Unfix a page/train from the buffer element.
  - Parameters
    - TrainID *trainId

      (IN) Page ID of the page or the first page of the train to be unfixed
    - Four type

      (IN) Type of the buffer
  - Return value
    - Four error code

– Example

```
Four edubtm_Fetch(
    PageID              *root,     /* IN: ID of the current root page of the subtree */
    ...)
{
    ...
    /* Unfix the page from the buffer */
    e = BfM_FreeTrain(root, PAGE_BUF);
    if (e < 0)  ERR(e);
    ...
}
```

- BfM_SetDirty()
  - Set the DIRTY bit to indicate that the page/train stored in the buffer element has been modified.
  - Parameters
    - TrainID *trainId

      (IN) Page ID of the page or the first page of the train whose DIRTY bit is to be set
    - Four type

      (IN) Type of the buffer
  - Return value
    - Four error code

– Example

```
Four edubtm_InitInternal(
    PageID  *internal,    /* IN: ID of the page to be initialized */
    ...)
{
    ...
    /* Set the DIRTY bit */
    e = BfM_SetDirty(internal, PAGE_BUF);
    if (e < 0) ERRB1(e, internal, PAGE_BUF);
    ...
}
```

- Util_getElementFromPool()
  - Allocate a new dealloc list element from the pool, and return the allocated element.
  - Parameters
    - Pool *aPool
      (IN) Element pool to be used for allocation
    - void *elem
      (OUT) Dealloc list element allocated
  - Return value
    - Four error code

– Example

```
Four edubtm_FreePages(...
    PageID              *curPid,     /* IN: ID of the page to be freed */
    Pool      *dlPool,        /* INOUT: pool of the elements of the dealloc list */
    DeallocListElem *dlHead)     /* INOUT: head of the dealloc list */
{
    DeallocListElem *dlElem;     /* pointer to the element of the dealloc list */
    ...
    /* Insert the deallocated page into the dealloc list */
    e = Util_getElementFromPool(dlPool, &dlElem);
    if (e < 0) ERR(e);

    dlElem->type = DL_PAGE;
    dlElem->elem.pid = *curPid;     /* ID of the deallocated page */
    dlElem->next = dlHead->next;
    dlHead->next = dlElem;
    ...
}
```

Given Functions

- btm_AllocPage()
  - Allocate a new page to be used as a B+ tree index page, and return the page ID of the allocated page.
  - Parameters
    - ObjectID *catObjForFile

      (IN) OID of the object that contains information (*sm_CatOverlayForSysTables*) about the index file containing the B+ tree index and about the data file indexed
    - PageID *nearPid

      (IN) Page ID of the page to be allocated or page ID of the page to which the page to be allocated should be physically close on the disk
    - PageID *newPid

      (OUT) Page ID of the page allocated
  - Return value
    - Four error code

– Example

```
Four EduBtM_CreateIndex(
    ObjectID *catObjForFile,     /* IN: ID of the object that contains the catalog
information */
    PageID *rootPid)     /* OUT: ID of the root page of the newly created B+tree */
{
    sm_CatOverlayForBtree *catEntry;     /* pointer to the B+tree file catalog
information */
    PhysicalFileID pFid;     /* ID of the first page in the file */
    ...
    MAKE_PHYSICALFILEID(pFid, catEntry->fid.volNo, catEntry->firstPage);

    /* Allocate a new page to be used as a B+ tree index page */
    e = btm_AllocPage(catObjForFile, (PageID *)&pFid, rootPid);
    if (e < 0)  ERR(e);
    ...
}
```


- btm_ObjectIdComp()
  - Compare two object IDs (OIDs) given by parameters, and return the comparison result.
  - Parameters
    - ObjectID *firstOid
      (IN) First object ID to be compared
    - ObjectID *secondOid
      (IN) Second object ID to be compared
  - Return value
    - Four EQUAL, GREATER, or LESS

– Example

```
Four edubtm_DeleteLeaf(...
    ObjectID                    *oid,    /* IN: ID of the object to be deleted */
    ...)
{
    ObjectID tOid;    /* ID of an object */
    ...
    /* Compare two object IDs */
    if(edubtm_ObjectIdComp(oid, &tOid) == EQUAL) {...}
    ...
}
```

- btm_root_delete()
  - Handle underflow that has occurred in the root page of a B+ tree index.
  - Parameters
    - PhysicalFileID *pFid
      (IN) File ID of the index file containing the B+ tree index (= page ID of the first page of the index file)
    - PageID *rootPid
      (IN) Page ID of the root page of the B+ tree index
    - Pool *dlPool
      (INOUT) Pool from which to allocate a new dealloc list element
    - DeallocListElem *dlHead
      (INOUT) Header pointing to the first element of the dealloc list
  - Return value
    - Four error code

– Example

```
Four EduBtM_DeleteObject(...
    PageID    *root,      /* IN: ID of the root page */
    ...
    Pool      *dlPool,        /* INOUT: pool of the elements of the dealloc list */
    DeallocListElem *dlHead)    /* INOUT: head of the dealloc list */
{
    Boolean lf;    /* TRUE if a page is not half full */
    PhysicalFileID pFid;    /* ID of the index file containing the B+ tree index */
    ...
    /* Handle underflow that has occurred in the root page of a B+ tree index */
    if(lf) {    // if underflow has occurred in the root page
        e = btm_root_delete(&pFid, root, dlPool, dlHead);
        if (e < 0) ERR(e);
    }
    ...
}
```

- btm_Underflow()
  - For the page of a B+ tree index that underflowed, merge or redistribute the page with the sibling page.
    - If two pages are merged, underflow of parent page can occur as the related index entry is deleted from the parent page.
    - If two pages are redistributed, overflow of parent page can occur as the related index entry is changed (existing index entry is deleted and new entry is inserted) in the parent page.
  - Parameters
    - PhysicalFileID *pFid
      (IN) File ID of the index file containing the B+ tree index (= page ID of the first page of the index file)
    - BtreePage *rpage
      (IN) Pointer to the buffer element containing parent page of the page that underflowed
    - PageID *child
      (IN) Page ID of the page that underflowed
    - Two slotNo
      (IN) Slot number of the slot containing the offset of the index entry pointing to the page that underflowed
    - Boolean *f
      (OUT) Flag indicating that underflow has occurred in the parent page
    - Boolean *h
      (OUT) Flag indicating that overflow has occurred in the parent page
    - InternalItem *item
      (OUT) Internal index entry that could not be inserted by the overflow of the parent page
    - Pool *dlPool
      (INOUT) Pool from which to allocate a new dealloc list element
    - DeallocListElem *dlHead
      (INOUT) Header pointing to the first element of the dealloc list
  - Return value
    - Four error code

– Example

```
Four edubtm_Delete(...
    PageID    *root,      /* IN: ID of the root page */
    ...
    Boolean                 *f,             /* OUT: whether the root page is half full */
    ...
    Pool      *dlPool,      /* INOUT: pool of the elements of the dealloc list */
    DeallocListElem *dlHead)    /* INOUT: head of the dealloc list */
{
    Boolean lf;    /* TRUE if a page is not half full */
    Boolean lh;    /* TRUE if a page is splitted */
    Two idx;    /* slot number */
    PageID child;    /* ID of the child page */
    BtreePage *rpage;    /* pointer to the root page */
    InternalItem litem;    /* Internal index entry */
    PhysicalFileID pFid;    /* ID of the index file containing the B+ tree index */
    ...
    /* Merge or redistribute the page with the sibling page */
    else if (lf) {    // if underflow occurs
        e = btm_Underflow(&pFid, rpage, &child, idx, f, &lh, &litem, dlPool, dlHead);
        if (e < 0) ERRB1(e, root, PAGE_BUF);
    }
    ...
}
```

Error Handling

- Error handling macro
  - ERR(e)
    - Write the error code *e* given as a parameter, the file name, and the position where the error occurred into the error log file (odysseus_error.log); return the error code.
    - Usage example

      ```
      if(root == NULL) ERR(eBADPARAMETER_BTM)
      ```
  - ERRB1(e, pid, t)
    - The same as ERR(e) except that it unfixes the page having the page ID, *pid*, given as the parameter before returning the error code *e*.
    - Usage example

      ```
      if(e < 0)   ERRB1(e, &newPid, PAGE_BUF)
      ```

- Error code

  See the $(EduBtM_HOME_DIR)/Header/EduBtM_errorcodes.h File.

How to Do the Project

- Files used in the project
  - Files that students are to implement
    - Skeleton file (.c file)

      Files containing the functions whose implementation part is omitted
  - Files given to students
    - Object file (.o file)

      File that ODYSSEUS/COSMOS, which is the underlying system, is compiled as an object file. It contains all the functions in ODYSSEUS/COSMOS storage system including the given lower-level functions that are called in the modules to be implemented.
    - Header file (.h file)

      Files containing the definition of data structures and function prototypes used in the modules to be implemented and the test module
    - Source code file of the test module

      Source code file of the test module to test the functions in the implemented module
    - Executable solution file

      Executable file showing a correct test result

- How to perform the project
  - Implement the functions in the skeleton files.
    - For the implementation, a variety of macros are available in the header files in the $(EduBtM_HOME_DIR)/Header directory.
  - Use the make command to compile the skeleton files implemented and link them with the given object file.
    - As a result of compiling and linking, an executable file is created to test functions of the modules implemented.
  - Compare the execution results of your executable file with those of the given executable solution file.

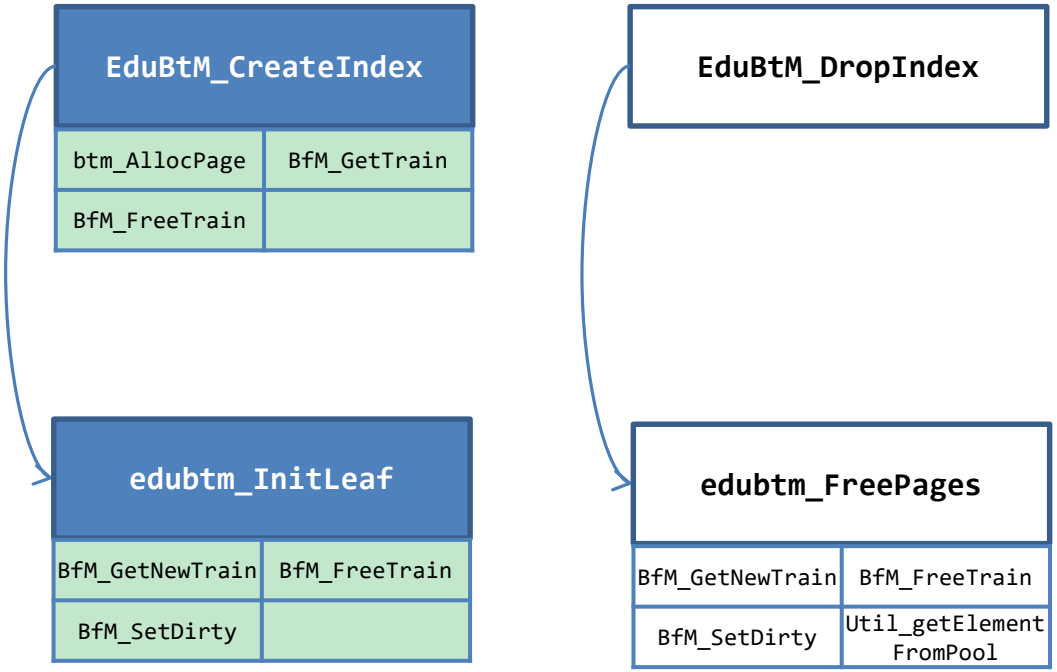
❖ How to test a function without implementing other functions
  ❖ In the file $(EduBtM_HOME_DIR)/Header/EduBtM_TestModule.h,
    ❖ For the API function that you have implemented, define the value of the corresponding macro to TRUE.
    ❖ For the API function that you have not implemented, define the value of the corresponding macro to FALSE.
  ❖ Enter the Make command to recompile the project.

❖ How to implement an API function without implementing some of its internal functions
  ❖ Use the default solution function (internal function name with the prefix "edu" omitted).
    ❖ e.g., the default solution function of the internal function edubtm_InitLeaf() is btm_InitLeaf().

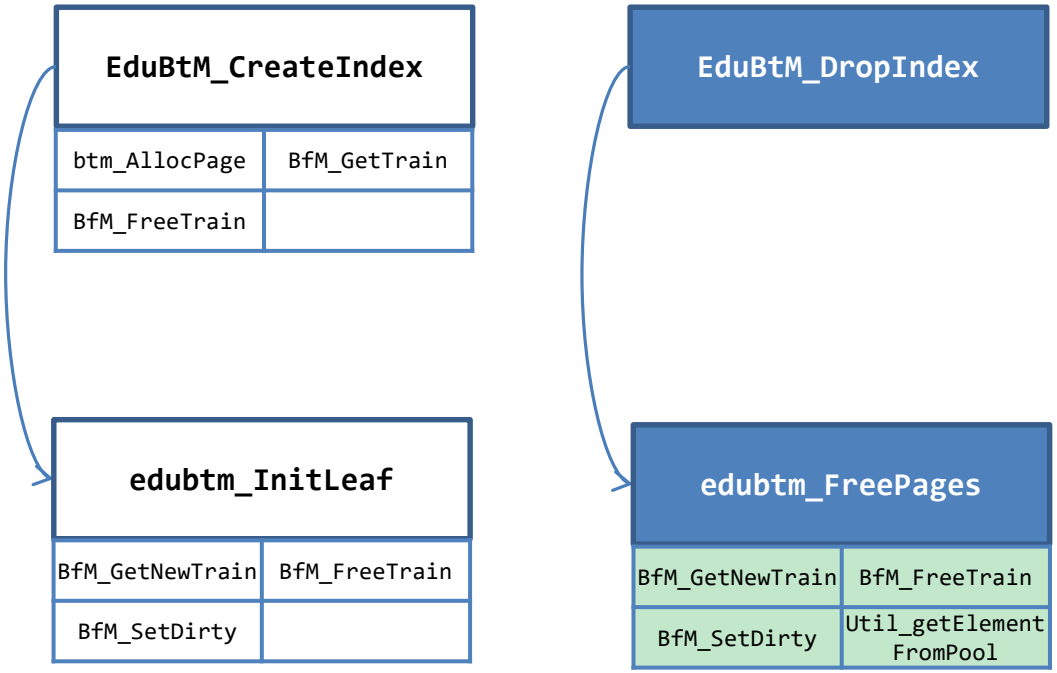
Appendix

Function Call Graph

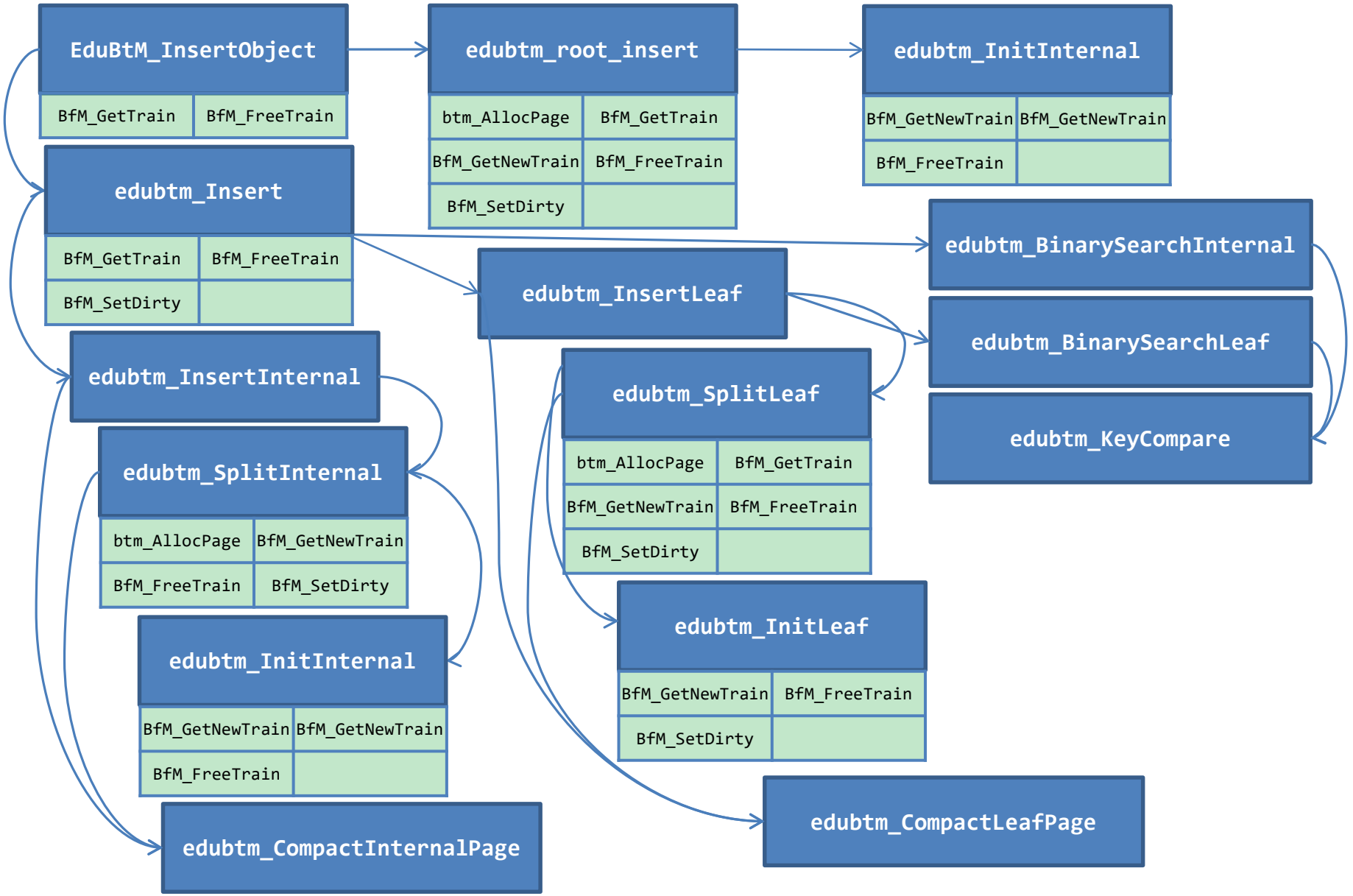
EduBtM_CreateIndex



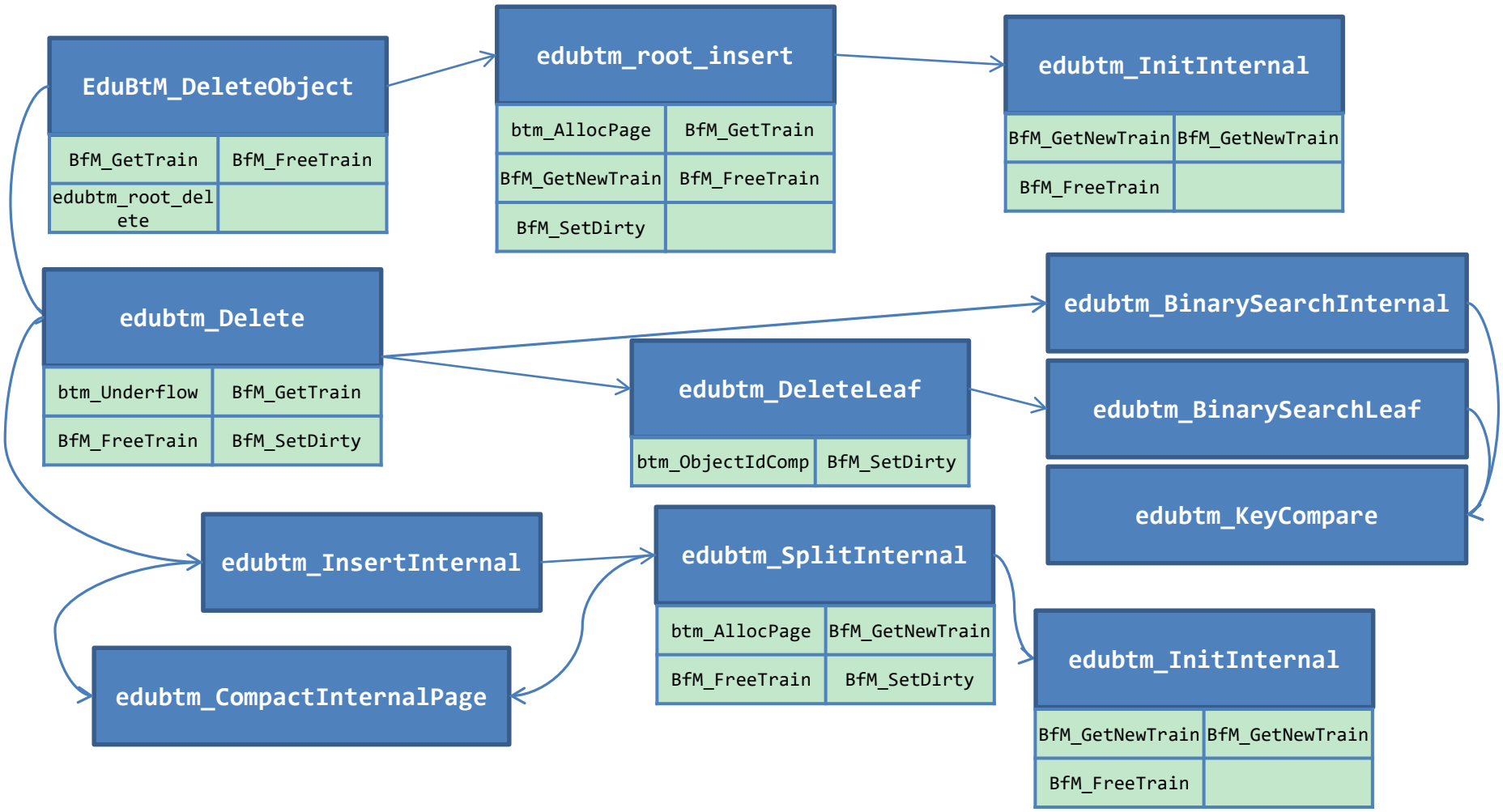
EduBtM_DropIndex



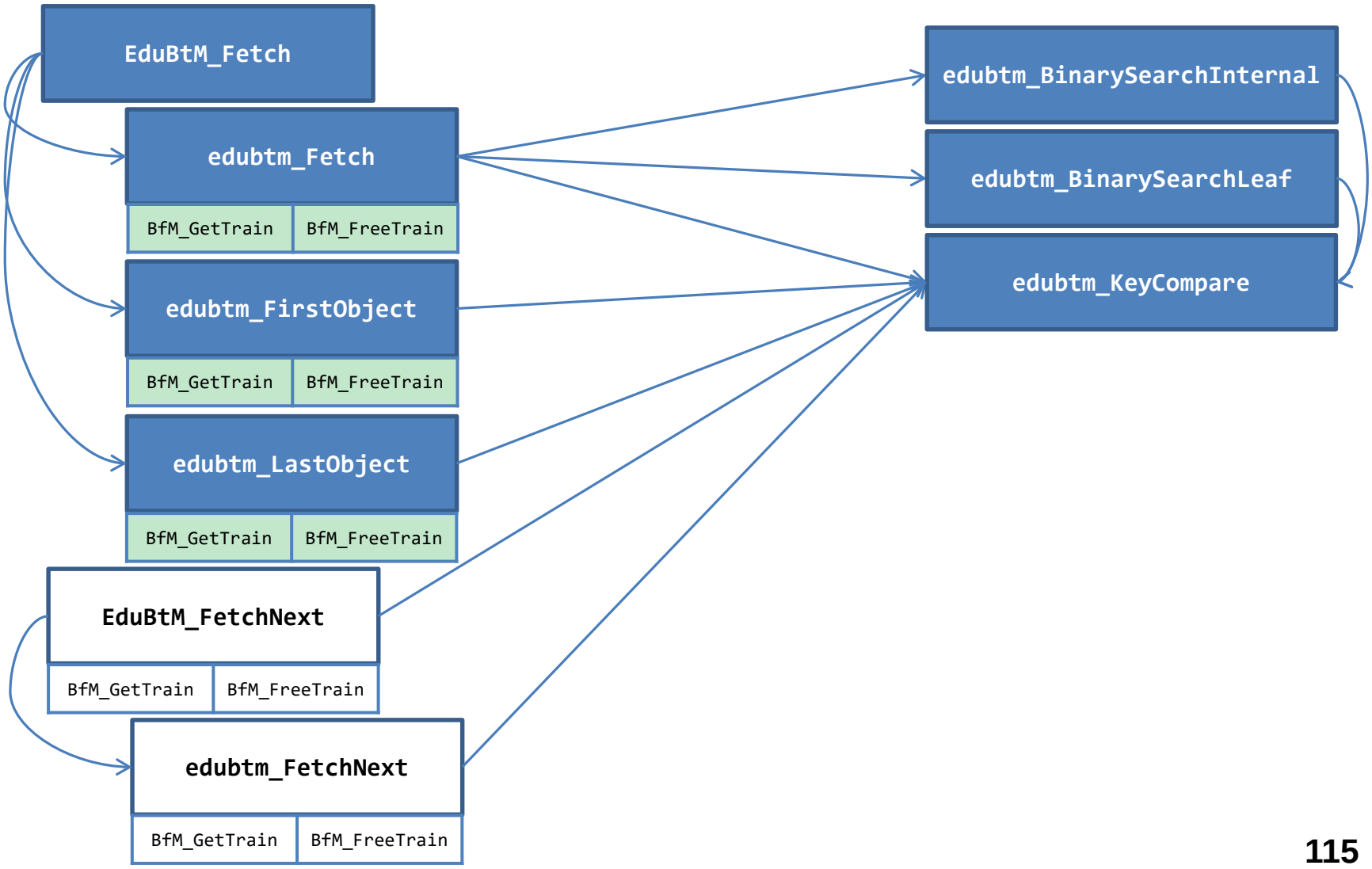
EduBtM_InsertObject



EduBtM_DeleteObject



EduBtM_Fetch



EduBtM_FetchNext