Kalashnikov DB

0.9.2

Generated by Doxygen 1.8.11

Contents

Chapter 1

Todo List

Member AK_acquire_lock (int, int, pthread_t)

Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

Member AK_acquire_lock (int, int, pthread_t)

Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

Member AK_archive_log ()

this function takes static filename to store the failed commands, create certain logic that would make the function to use dynamic filename (this is partly implemented inside AK_get_timestamp, but there is no logic that uses the last file when recovering - recovery.c) {link} recovery.c function test

Member AK_execute_commands (command *, int)

Check multithreading, check if it's working correctly

Member AK_execute_commands (command *, int)

Check multithreading, check if it's working correctly

Member AK_get_timestamp ()

Think about this in the future when creating multiple binary recovery files. Implementation gives the timestamp, but is not used anywhere for now.

Member AK_memory_block_hash (int)

The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

Member AK_memory_block_hash (int)

The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 _file_metadata Struct Reference

Public Attributes

- char * **new_path**
- char * **new_name**
- char * **old_path**
- char * **old_name**
- char * **checksum**

The documentation for this struct was generated from the following file:

- file/blobs.h

4.2 AK_agg_input Struct Reference

Structure that contains attributes from table header, tasks for this table and counter value.

`#include <aggregation.h>`

Collaboration diagram for AK_agg_input:

Public Attributes

- AK_header **attributes** [MAX_ATTRIBUTES]
- int **tasks** [MAX_ATTRIBUTES]
- int **counter**

4.2.1 Detailed Description

Structure that contains attributes from table header, tasks for this table and counter value.

Author

Unknown

The documentation for this struct was generated from the following file:

- rel/aggregation.h

4.3 AK_agg_value Struct Reference

Structure that contains atribute name, date and aggregation task associated.

`#include <aggregation.h>`

Public Attributes

- char **att_name** [MAX_ATT_NAME]
- char **data** [MAX_VARCHAR_LENGTH]
- int **agg_task**

4.3.1 Detailed Description

Structure that contains atribute name, date and aggregation task associated.

Author

Unknown

The documentation for this struct was generated from the following file:

- rel/aggregation.h

4.4 AK_block Struct Reference

Structure that defines a block of data inside a DB file. It contains address, type, chained_with, AK_free space, last_tuple_dict_id, header and tuple_dict and data.

`#include <dbman.h>`

Collaboration diagram for AK_block:

Public Attributes

- int address

  block number (address) in DB file
- int type

  block type (can be BLOCK_TYPE_FREE, BLOCK_TYPE_NORMAL or BLOCK_TYPE_CHAINED)
- int chained_with

  address of chained block; NOT_CHAINED otherwise
- int AK_free_space

  AK_free space in block.
- int **last_tuple_dict_id**
- AK_header header [MAX_ATTRIBUTES]

  attribute definitions
- AK_tuple_dict tuple_dict [DATA_BLOCK_SIZE]

  dictionary of data entries
- unsigned char data [DATA_BLOCK_SIZE ∗DATA_ENTRY_SIZE]

  actual data entries

4.4.1 Detailed Description

Structure that defines a block of data inside a DB file. It contains address, type, chained_with, AK_free space, last_tuple_dict_id, header and tuple_dict and data.

Author

Markus Schatten

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.5 AK_block_activity Struct Reference

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked↩ _for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and uninteruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

```
#include <dbman.h>
```

Public Attributes

- short **locked_for_reading**
- short **locked_for_writing**
- pthread_mutex_t **block_lock**
- pthread_cond_t **writing_done**
- pthread_cond_t **reading_done**
- int * **thread_holding_lock**

4.5.1 Detailed Description

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked↩ _for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and uninteruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

Author

Domagoj Šitum

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.6 AK_blocktable Struct Reference

Public Attributes

- unsigned int **allocationtable** [DB_FILE_BLOCKS_NUM_EX]
- unsigned char **bittable** [BITNSLOTS(DB_FILE_BLOCKS_NUM_EX)]
- int **last_allocated**
- int **last_initialized**
- int **prepared**
- time_t **ltime**

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.7 AK_command_recovery_struct Struct Reference

recovery structure used to recover commands from binary file

`#include <memoman.h>`

Public Attributes

- int **operation**
- char * **table_name**
- char ** **arguments**

4.7.1 Detailed Description

recovery structure used to recover commands from binary file

Structure that contains all vital information for the command that is about to execute. It is defined by the operation (INSERT, UPDATE, DELETE that are defined inside the const.c file), table where the data is stored, and certain data that will be stored.

Author

Tomislav Turek

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.8 AK_command_struct Struct Reference

Public Attributes

- int **id_command**
- char * **tblName**
- void * **parameters**

The documentation for this struct was generated from the following file:

- sql/command.h

4.9 AK_create_table_struct Struct Reference

Public Attributes

- char **name** [MAX_ATT_NAME]
- int **type**

The documentation for this struct was generated from the following file:

- file/table.h

4.10 AK_db_cache Struct Reference

Structure that defines global cache memory.

`#include <memoman.h>`

Collaboration diagram for AK_db_cache:

Public Attributes

- AK_mem_block * cache [MAX_CACHE_MEMORY]
  last recently read blocks
- int next_replace
  next cached block to be replaced (0 - MAX_CACHE_MEMORY-1); depends on caching algorithm

4.10.1 Detailed Description

Structure that defines global cache memory.

Author

Unknown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.11 AK_header Struct Reference

Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.

`#include <dbman.h>`

Public Attributes

- int type
  type of attribute
- char att_name [MAX_ATT_NAME]
  attribute name
- int integrity [MAX_CONSTRAINTS]
  standard integrity costraints
- char constr_name [MAX_CONSTRAINTS][MAX_CONSTR_NAME]
  extra integrity constraint names
- char constr_code [MAX_CONSTRAINTS][MAX_CONSTR_CODE]
  extra integrity costraint codes

4.11.1 Detailed Description

Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.

Author

Markus Schatten

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.12 AK_mem_block Struct Reference

Structure that defines a block of data in memory.

`#include <memoman.h>`

Collaboration diagram for AK_mem_block:

Public Attributes

- AK_block * block
  pointer to block from DB file
- int dirty
  dirty bit (BLOCK_CLEAN if unchanged; BLOCK_DIRTY if changed but not yet written to file)
- unsigned long timestamp_read
  timestamp when the block has lastly been read
- unsigned long timestamp_last_change
  timestamp when the block has lastly been changed

4.12.1 Detailed Description

Structure that defines a block of data in memory.

Author

Unknown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.13 AK_query_mem Struct Reference

Structure that defines global query memory.

`#include <memoman.h>`

Collaboration diagram for AK_query_mem:

Public Attributes

- AK_query_mem_lib * parsed
  parsed queries
- AK_query_mem_dict * dictionary
  obtained data dictionaries
- AK_query_mem_result * result
  obtained query results

4.13.1 Detailed Description

Structure that defines global query memory.

Author

Unknown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.14 AK_query_mem_dict Struct Reference

Structure that defines global query memory for data dictionaries.

`#include <memoman.h>`

Collaboration diagram for AK_query_mem_dict:

Public Attributes

- AK_tuple_dict * dictionary [MAX_QUERY_DICT_MEMORY]
  last used data dictionaries
- int next_replace
  next dictionary to be replaced (0 - MAX_QUERY_DICT_MEMORY-1); field pointer (LIFO)

4.14.1 Detailed Description

Structure that defines global query memory for data dictionaries.

Author

Unkown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.15 AK_query_mem_lib Struct Reference

Structure that defines global query memory for libraries.

`#include <memoman.h>`

Public Attributes

- char parsed [MAX_QUERY_LIB_MEMORY]
  last parsed queries; to be changed to more adequate data structure
- int next_replace
  next query to be replaced (0 - MAX_QUERY_LIB_MEMORY-1); field pointer (LIFO)

4.15.1 Detailed Description

Structure that defines global query memory for libraries.

Author

Unkown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.16 AK_query_mem_result Struct Reference

Structure that defines global query memory for results.

`#include <memoman.h>`

Collaboration diagram for AK_query_mem_result:

Public Attributes

- AK_results * **results**
- int next_replace

  next result to be replaced (0 - MAX_QUERY_RESULT_MEMORY-1); field pointer (LIFO)

4.16.1 Detailed Description

Structure that defines global query memory for results.

Author

Unknown

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.17 AK_redo_log Struct Reference

Structure that defines global redo log.

`#include <memoman.h>`

Collaboration diagram for AK_redo_log:

Public Attributes

- AK_command_recovery_struct * **command_recovery**
- int **number**

4.17.1 Detailed Description

Structure that defines global redo log.

The structure defines an array of commands being executed at the moment. If and when commands fail to execute, the rest of the commands that did not execute will be stored inside a binary file and the system will try recovery and execution for those commands. With the array, we also store a number that defines the number of commands that failed to execute (length of command_recovery array).

Author

Dražen Bandić, updated by Tomislav Turek

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.18 AK_ref_item Struct Reference

Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

`#include <reference.h>`

Public Attributes

- char **table** [MAX_ATT_NAME]
- char **attributes** [MAX_REFERENCE_ATTRIBUTES][MAX_ATT_NAME]
- char **parent** [MAX_ATT_NAME]
- char **parent_attributes** [MAX_REFERENCE_ATTRIBUTES][MAX_ATT_NAME]
- int **attributes_number**
- char **constraint** [MAX_VARCHAR_LENGTH]
- int **type**

4.18.1 Detailed Description

Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

Author

Dejan Franković

The documentation for this struct was generated from the following file:

- sql/cs/reference.h

4.19 AK_results Struct Reference

Structure used for in-memory result caching.

`#include <memoman.h>`

Collaboration diagram for AK_results:

Public Attributes

- unsigned long **result_id**
- int **result_size**
- char **date_created** [80]
- short **free**
- char * **source_table**
- AK_block * **result_block**
- AK_header **header** [MAX_ATTRIBUTES]

4.19.1 Detailed Description

Structure used for in-memory result caching.

Author

Mario Novoselec

The documentation for this struct was generated from the following file:

- mm/memoman.h

4.20 AK_tuple_dict Struct Reference

Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.

`#include <dbman.h>`

Public Attributes

- int type
  data entry type
- int address
  data entry address (in AK_block->data)
- int size
  *data entry size (using sizeof(***))*

4.20.1 Detailed Description

Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.

Author

Markus Schatten

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.21 blocktable Struct Reference

Structure that defines bit status of blocks, last initialized and last allocated index.

`#include <dbman.h>`

4.21.1 Detailed Description

Structure that defines bit status of blocks, last initialized and last allocated index.

Author

dv

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.22 btree_node Struct Reference

Collaboration diagram for btree_node:

Public Attributes

- int **values** [B]
- struct_add **pointers** [B+1]

The documentation for this struct was generated from the following file:

- file/idx/btree.h

4.23 bucket_elem Struct Reference

Structure for defining a single bucket element.

`#include <hash.h>`

Collaboration diagram for bucket_elem:

Public Attributes

- unsigned int value

  bucket element hash value
- struct_add add

  bucket element address values

4.23.1 Detailed Description

Structure for defining a single bucket element.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/idx/hash.h

4.24 cost_eval_t Struct Reference

Stucture for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)

`#include <rel_eq_assoc.h>`

Public Attributes

- int **value**
- char **data** [MAX_VARCHAR_LENGTH]

4.24.1 Detailed Description

Stucture for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)

Author

Dino Laktašić

The documentation for this struct was generated from the following file:

- opti/rel_eq_assoc.h

4.25 drop_arguments Struct Reference

Collaboration diagram for drop_arguments:

Public Attributes

- void * **value**
- struct drop_arguments * **next**

The documentation for this struct was generated from the following file:

- sql/drop.h

4.26 hash_bucket Struct Reference

Structure for hash bucket for table hashing.

`#include <hash.h>`

Collaboration diagram for hash_bucket:

Public Attributes

- int bucket_level

  hash bucket level
- bucket_elem element [HASH_BUCKET_SIZE]

  hash bucket array of bucket_elem elements

4.26.1 Detailed Description

Structure for hash bucket for table hashing.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/idx/hash.h

4.27 hash_info Struct Reference

Structure for defining a hash info element.

`#include <hash.h>`

Public Attributes

- int modulo
  modulo value for hash function
- int main_bucket_num
  bucket number
- int hash_bucket_num
  hash bucket number

4.27.1 Detailed Description

Structure for defining a hash info element.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/idx/hash.h

4.28 intersect_attr Struct Reference

Structure defines intersect attribute.

`#include <intersect.h>`

Public Attributes

- int type
  type of attribute
- char att_name [MAX_ATT_NAME]
  attribute name

4.28.1 Detailed Description

Structure defines intersect attribute.

Author

Dino Laktašić

The documentation for this struct was generated from the following file:

- rel/intersect.h

4.29 list_structure_ad Struct Reference

Collaboration diagram for list_structure_ad:

Public Attributes

- char * attName
  attribute name
- struct_add add
  addresses
- struct list_structure_ad * next
  next node pointer

The documentation for this struct was generated from the following file:

- file/idx/index.h

4.30 list_structure_add Struct Reference

Structure that defines linked list node for index.

`#include <index.h>`

4.30.1 Detailed Description

Structure that defines linked list node for index.

The documentation for this struct was generated from the following file:

- file/idx/index.h

4.31 main_bucket Struct Reference

Structure for defining main bucket for table hashing.

`#include <hash.h>`

Collaboration diagram for main_bucket:

Public Attributes

- bucket_elem element [MAIN_BUCKET_SIZE]
  main bucket array of bucket_elem elements

4.31.1 Detailed Description

Structure for defining main bucket for table hashing.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/idx/hash.h

4.32 memoryAddresses Struct Reference

Structure that represents a linked list of locked addresses.

`#include <transaction.h>`

Collaboration diagram for memoryAddresses:

Public Attributes

- int **adresa**
- struct memoryAddresses * **nextElement**

4.32.1 Detailed Description

Structure that represents a linked list of locked addresses.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.33 observable_transaction Struct Reference

Structure which defines transaction observable type.

`#include <transaction.h>`

4.33.1 Detailed Description

Structure which defines transaction observable type.

Author

Ivan Pusic

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.34 observable_transaction_struct Struct Reference

Public Attributes

- int(∗ **AK_transaction_register_observer**)(struct observable_transaction_struct ∗, AK_observer ∗)
- int(∗ **AK_transaction_unregister_observer**)(struct observable_transaction_struct ∗, AK_observer ∗)
- void(∗ **AK_lock_released**)()
- void(∗ **AK_transaction_finished**)()
- void(∗ **AK_all_transactions_finished**)()
- AK_observable ∗ **observable**

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.35 observer_lock Struct Reference

Structure which defines transaction lock observer type.

`#include <transaction.h>`

Public Attributes

- AK_observer ∗ **observer**

4.35.1 Detailed Description

Structure which defines transaction lock observer type.

Author

Ivan Pusic

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.36 root_info Struct Reference

Public Attributes

- int **root**
- int **level** [ORDER]

The documentation for this struct was generated from the following file:

- file/idx/btree.h

4.37 search_params Struct Reference

Structure that contains attribute name, lower and upper data value, special(NULL or ∗) which is input for AK_↩ equisearch_unsorted and AK_rangesearch_unsorted.

`#include <filesearch.h>`

Public Attributes

- char ∗ szAttribute

  name of attribute
- void ∗ pData_lower

  pointer to lower value of search range
- void ∗ pData_upper

  pointer to upper value of search range
- int iSearchType

  if searching for NULL values, set to SEARCH_NULL, all values -> SEARCH_ALL, particular value -> SEARCH_↩ PARTICULAR, range of values -> SEARCH_RANGE

4.37.1 Detailed Description

Structure that contains attribute name, lower and upper data value, special(NULL or ∗) which is input for AK_↩ equisearch_unsorted and AK_rangesearch_unsorted.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/filesearch.h

4.38 search_result Struct Reference

Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted.

`#include <filesearch.h>`

Public Attributes

- int * aiTuple_addresses
  array of tuple addresses
- int * aiBlocks
  array of blocks to which the tuple addresses are relative
- int iNum_tuple_addresses
  number of tuple addresses/blocks in corresponding arrays
- int * aiSearch_attributes
  array of indexes of searched-for attributes
- int iNum_search_attributes
  number of searched-for attributes in array
- int iNum_tuple_attributes
  number of attributes in tuple

4.38.1 Detailed Description

Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/filesearch.h

4.39 struct_add Struct Reference

Structure defining node address.

`#include <index.h>`

Public Attributes

- int addBlock
  block address
- int indexTd
  index table destination

4.39.1 Detailed Description

Structure defining node address.

Author

Unknown

The documentation for this struct was generated from the following file:

- file/idx/index.h

4.40 table_addresses Struct Reference

Structure that defines start and end address of extent.

`#include <dbman.h>`

Public Attributes

- int address_from [MAX_EXTENTS_IN_SEGMENT]
  sturcture for extents start end stop adresses
- int **address_to** [MAX_EXTENTS_IN_SEGMENT]

4.40.1 Detailed Description

Structure that defines start and end address of extent.

Author

Matija Novak

The documentation for this struct was generated from the following file:

- dm/dbman.h

4.41 threadContainer Struct Reference

Structure that represents a linked list of threads.

`#include <transaction.h>`

Collaboration diagram for threadContainer:

Public Attributes

- pthread_t **thread**
- struct threadContainer * **nextThread**

4.41.1 Detailed Description

Structure that represents a linked list of threads.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.42 transaction_list_elem Struct Reference

Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table.

`#include <transaction.h>`

Collaboration diagram for transaction_list_elem:

Public Attributes

- int **address**
- int **lock_type**
- int **isWaiting**
- struct transaction_locks_list_elem * **DLLLocksHead**
- struct transaction_list_elem * **nextBucket**
- struct transaction_list_elem * **prevBucket**
- AK_observer_lock * **observer_lock**

4.42.1 Detailed Description

Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.43 transaction_list_head Struct Reference

Structure that represents LockTable entry about doubly linked list of collision in Hash table.

`#include <transaction.h>`

Collaboration diagram for transaction_list_head:

Public Attributes

- struct transaction_list_elem * **DLLHead**

4.43.1 Detailed Description

Structure that represents LockTable entry about doubly linked list of collision in Hash table.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.44 transaction_locks_list_elem Struct Reference

Structure that represents LockTable entry about transaction resource lock.

`#include <transaction.h>`

Collaboration diagram for transaction_locks_list_elem:

Public Attributes

- pthread_t **TransactionId**
- int **lock_type**
- int **isWaiting**
- struct transaction_locks_list_elem * **nextLock**
- struct transaction_locks_list_elem * **prevLock**

4.44.1 Detailed Description

Structure that represents LockTable entry about transaction resource lock.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

4.45 transactionData Struct Reference

Structure used to transport transaction data to the thread.

`#include <transaction.h>`

Collaboration diagram for transactionData:

Public Attributes

- int **lengthOfArray**
- command * **array**

4.45.1 Detailed Description

Structure used to transport transaction data to the thread.

Author

Frane Jakelić

The documentation for this struct was generated from the following file:

- trans/transaction.h

Chapter 5

File Documentation

5.1 dm/dbman.c File Reference

```
#include "dbman.h"
```
Include dependency graph for dbman.c:

5.2 dm/dbman.h File Reference

```
#include "../auxi/auxiliary.h"
#include <errno.h>
#include <pthread.h>
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../auxi/mempro.h"
#include <limits.h>
```
Include dependency graph for dbman.h: This graph shows which files directly or indirectly include this file:

Classes

- struct AK_header

  Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.
- struct AK_tuple_dict

  Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.
- struct AK_block

  Structure that defines a block of data inside a DB file. It contains address, type, chained_with, AK_free space, last_tuple_dict_id, header and tuple_dict and data.
- struct table_addresses

  Structure that defines start and end address of extent.
- struct AK_blocktable
- struct AK_block_activity

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked_for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and uninteruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

Macros

- #define **BITMASK**(b) (1 << ((b) % CHAR_BIT))
- #define **BITSLOT**(b) ((int)((b) / CHAR_BIT))
- #define **BITSET**(a, b) ((a)[BITSLOT(b)] |= BITMASK(b))
- #define **BITCLEAR**(a, b) ((a)[BITSLOT(b)] &= ~BITMASK(b))
- #define **BITTEST**(a, b) ((a)[BITSLOT(b)] & BITMASK(b))
- #define **BITNSLOTS**(nb) ((int)(nb + CHAR_BIT - 1) / CHAR_BIT)
- #define **SEGMENTLENGTH**() (BITNSLOTS(DB_FILE_BLOCKS_NUM) + 2*sizeof(int))
- #define **DB_FILE_SIZE_EX** 40
- #define **DB_FILE_BLOCKS_NUM_EX** (int)(1024 * 1024 * DB_FILE_SIZE_EX / sizeof(AK_block))
- #define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)

  Holds size of allocation table.
- #define CHAR_IN_LINE 80

  How many characters could line contain.
- #define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY

  How many blocks would be initially allocated.

Enumerations

- enum AK_allocation_set_mode {
  allocationSEQUENCE = 10001, **allocationUPPER**, **allocationLOWER**, **allocationAROUND**, **allocationNOMODE** }

  Different modes to obtain allocation indexes: SEQUENCE - first found set of sequence indexes UPPER - set tries to place itself to upper part od allocation table LOWER - set tries to place itself to lower part od allocation table AROUND - set tries to place itself around targeted index.

Functions

- int AK_print_block (AK_block *block, int num, char *gg, FILE *fpp)

  Function that dumps block.
- void **AK_allocationbit_test** ()
- void **AK_allocationtable_test** ()
- int * AK_increase_extent (int start_address, int add_size, AK_allocation_set_mode *mode, int border, int target, AK_header *header, int gl)

  Function alocates new blocks for increasing extent size.
- int * AK_get_extent (int start_address, int desired_size, AK_allocation_set_mode *mode, int border, int target, AK_header *header, int gl)

  Function alocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.
- int * AK_get_allocation_set (int *bitsetbs, int fromWhere, int gaplength, int num, AK_allocation_set_mode mode, int target)

  Function prepare demanded sets from allocation table.

- int AK_copy_header (AK_header *header, int *blocknum, int num)

  Function copy header to blocks. Completely thread-safe.
- int AK_allocate_blocks (FILE *db, AK_block *block, int FromWhere, int HowMany)

  Function that allocates new blocks by placing them to appropriate place and then update last initialized index.
- AK_block * AK_init_block ()

  Function that initializes new block.
- void AK_allocationtable_dump (int zz)

  Function dumpes allocation table.
- void AK_blocktable_dump (int zz)

  Function dumpes allocation table.
- int AK_blocktable_flush ()

  Function flushes bitmask table to disk.
- void AK_thread_safe_block_access_test ()

  This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.
- void * AK_read_block_for_testing (void *address)

  This function is only for testing. It has to be there, because pthread_create only accepts void function_name (void *) function format. So AK_read_block is no-go for pthread_create.*
- void * AK_write_block_for_testing (void *block)

  This function is only for testing. It has to be there, because pthread_create only accepts void function_name (void *) function format. So AK_write_block is no-go for pthread_create.*
- int AK_blocktable_get ()

  Function gets allocation table from disk.
- int fsize (FILE *fp)

  Helper function to determine file size.
- int AK_init_allocation_table ()

  Function that initializes allocation table, write it to disk and cache in memory.
- int AK_init_db_file (int size)

  Function that initializes a new database file named DB_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE_INT, attribute names are set to FREE_CHAR, integrities are set to FREE_INT, constraint names are set to FREE_CHAR, constraint names and codes are set to FREE_CHAR. Type, address and size of tuples are set to FREE_INT. Data in block is set to FREE_CHAR. Type of block is BLOCK_TYPE_FREE, it is not chained and id of last tuple is 0.
- AK_block * AK_read_block (int address)

  Function that reads a block at a given address (block number less than db_file_size). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.
- int AK_write_block (AK_block *block)

  Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.
- int AK_new_extent (int start_address, int old_size, int extent_type, AK_header *header)

  Function alocates new extent of blocks. If argument "old_size" is 0 than size of extent is INITIAL_EXTENT_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is incremented.
- int AK_new_segment (char *name, int type, AK_header *header)

  Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL_EXTENT_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL_EXTENT_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK_free.
- AK_header * AK_create_header (char *name, int type, int integrity, char *constr_name, char *contr_code)

  Function for creating header and initalize integrity, constraint name and constraint code with parameter values of function.
- void AK_insert_entry (AK_block *block_address, int type, void *entry_data, int i)

Function for inserting entry in tuple_dict and data of a block. Address, type and size of catalog_tuple_dict are set. Free space of block is also set.

- int AK_init_system_tables_catalog (int relation, int attribute, int index, int view, int sequence, int function, int function_arguments, int trigger, int trigger_conditions, int db, int db_obj, int user, int group, int user_group, int user_right, int group_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

  Function initialises the sytem table catalog and writes the result in first (0) block in db_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained_with and AK_free_space attributes are initialized. Names of various database elements are written in block.
- void AK_memset_int (void *block, int value, size_t num)

  Function that sets the first num ints of a block of memory to the specified value.
- int AK_register_system_tables (int relation, int attribute, int index, int view, int sequence, int function, int function_arguments, int trigger, int trigger_conditions, int db, int db_obj, int user, int group, int user_group, int user_right, int group_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

  Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.
- int AK_init_system_catalog ()

  Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function AK_register_system_tables() to register system tables.
- int AK_delete_block (int address)

  Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK_free" values. In tuple dictionary type, address and size are set to FREE_INT values. Data of block is set to FREE_CHAR.
- int AK_delete_extent (int begin, int end)

  Function deletes an extent between begin and end blocks.
- int AK_delete_segment (char *name, int type)
- int AK_init_disk_manager ()

Variables

- FILE * db

  Variable that defines the DB file file handle.
- unsigned int db_file_size

  Variable that defines the size of the DB file (in blocks)
- AK_blocktable * AK_allocationbit

  Global variable that holds allocation bit-vector.
- AK_block_activity * **AK_block_activity_info**
- AK_synchronization_info * **dbmanFileLock**

5.2.1 Detailed Description

Header file that defines includes and datastructures for the disk manager of Kalashnikov DB

5.2.2 Macro Definition Documentation

5.2.2.1 #define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)

Holds size of allocation table.

Author

dv

5.2.2.2 #define CHAR_IN_LINE 80

How many characters could line contain.

Author

dv

5.2.2.3 #define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY

How many blocks would be initially allocated.

Author

dv

5.2.3 Enumeration Type Documentation

5.2.3.1 enum AK_allocation_set_mode

Different modes to obtain allocation indexes: SEQUENCE - first found set of sequence indexes UPPER - set tries to place itself to upper part od allocation table LOWER - set tries to place itself to lower part od allocation table AROUND - set tries to place itself around targeted index.

Author

dv

5.2.4 Function Documentation

5.2.4.1 int AK_allocate_blocks (FILE * *db,* AK_block * *block,* int *FromWhere,* int *HowMany*)

Function that allocates new blocks by placing them to appropriate place and then update last initialized index.

Author

Markus Schatten , rearranged by dv

Returns

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

5.2.4.2 void AK_allocationtable_dump (int *zz*)

Function dumpes allocation table.

Author

dv

Returns

nothing

5.2.4.3 void AK_blocktable_dump (int *zz*)

Function dumpes allocation table.

Author

dv

Returns

nothing

5.2.4.4 int AK_blocktable_flush ( )

Function flushes bitmask table to disk.

Author

dv

Returns

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

5.2.4.5 int AK_blocktable_get ( )

Function gets allocation table from disk.

Author

dv

Returns

EXIT_SUCCESS if the file has been taken from disk, EXIT_ERROR otherwise

5.2.4.6 int AK_copy_header (AK_header * *header,* int * *blocknum,* int *num*)

Function copy header to blocks. Completely thread-safe.

Author

Nikola Bakoš, updated by Dino Laktašiæ (fixed header BUG), refurbished by dv

Parameters

header	pointer to header provided for copy
blocknum	pointer to addresses of blocks that header needs to be copied
num	number of blocks waiting for its header

Returns

number of performed header copy

var to check the number of written blocks

if write of block succeded increase var num_blocks, else nothing

5.2.4.7 AK_header* AK_create_header (char * *name*, int *type*, int *integrity*, char * *constr_name*, char * *contr_code*)

Function for creating header and initalize integrity, constraint name and constraint code with parameter values of function.

Author

Matija Novak

Parameters

name	name of the atribute
type	type of the atribute
integrity	standard integrity costraint
constr_name	extra integrity constraint name
contr_code	extra integrity costraint code

Returns

AK_header

initialize catalog_header->integrity and catalog_header->constr_name[][] and catalog_header->constr_code[][] with data given as functions parameters

5.2.4.8 int AK_delete_block (int *address*)

Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK_free" values. In tuple dictionary type, address and size are set to FREE_INT values. Data of block is set to FREE_CHAR.

Author

Markus Schatten

Parameters

address	address of the block to be deleted

Returns

returns EXIT_SUCCESS if deletion successful, else EXIT_ERROR

5.2.4.9 int AK_delete_extent (int *begin,* int *end*)

Function deletes an extent between begin and end blocks.

Author

Dejan Samboliæ

Parameters

begin	address of extent's first block
end	address of extent's last block

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.2.4.10 int AK_delete_segment (char * *name,* int *type*)

Author

Mislav Èakariæ

Parameters

name	name of the segment
type	type of the segment

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.2.4.11 int* AK_get_allocation_set (int * *bitsetbs,* int *fromWhere,* int *gaplength,* int *num,* AK_allocation_set_mode *mode,* int *target*)

Function prepare demanded sets from allocation table.

Author

dv

Parameters

bitset	int pointer, cointainer for bit set
fromWhere	has meaning just in SEQUENCE case. It describes from which address searching have to start.
gaplength	tells how many used blocks could be tolerated in bitset
num	Tells how many AK_free blocks has been needed
mode	Defines how to obtain set of indexes to AK_free addresses
target	has meaning just in case mode=AROUND: set must be as much as possible close to target from both sides

Returns

pointer to integer indexes field with prepared set. If it , for any reason, is not possible set has FREE_INT fullfilment.

5.2.4.12 int* AK_get_extent (int *start_address,* int *desired_size,* AK_allocation_set_mode * *mode,* int *border,* int *target,* AK_header * *header,* int *gl*)

Function alocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

Author

dv

Parameters

start_address	address (block number) to start searching for sufficient space
desired_size	number of desired blocks
AK_allocation_set_mode	a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
border	number of allocated blocks gap
target	block address around which other blocks have to be searched
header	pointer to header that should be written to the new extent (all blocks)
int	gl gap size

Returns

pointer to set of alocated block addresses

vars for loop [for]

if some blocks are not succesfully allocated, which means that the extend allocation has FAILED

5.2.4.13 int* AK_increase_extent (int *start_address,* int *add_size,* AK_allocation_set_mode * *mode,* int *border,* int *target,* AK_header * *header,* int *gl*)

Function alocates new blocks for increasing extent size.

Author

dv

Parameters

start_address	first address of extent that is subject of increasing
add_size	number how many new blocks is to be added to existing extent
AK_allocation_set_mode	a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
border	number of allocated blocks gap
target	block address around which other blocks have to be searched
header	pointer to header that should be written to the new extent (all blocks)
int	gl gap size

Returns

pointer to set of alocated block addresses

vars for loop [for]

5.2.4.14 int AK_init_allocation_table ( )

Function that initializes allocation table, write it to disk and cache in memory.

Author

dv

Returns

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

5.2.4.15 AK_block* AK_init_block ( )

Function that initializes new block.

Author

Markus Schatten , rearranged by dv

Returns

pointer to block allocated in memory

5.2.4.16 int AK_init_db_file (int *size*)

Function that initializes a new database file named DB_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE_INT, attribute names are set to FREE_CHAR, integrities are set to FREE_INT, constraint names are set to FREE_CHAR, constraint names and codes are set to FREE_CHAR. Type, address and size of tuples are set to FREE_INT. Data in block is set to FREE_CHAR. Type of block is BLOCK_TYPE_FREE, it is not chained and id of last tuple is 0.

Author

Markus Schatten

Parameters

size	size of new file in in blocks

Returns

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

5.2.4.17 int AK_init_disk_manager ( )

Author

Markus Schatten

Returns

Function that calls functions AK_init_db_file() and AK_init_system_catalog() to initialize disk manager. It also calls AK_allocate_array_currently_accessed_blocks() to allocate memory needed for thread-safe reading and writing to disk.

5.2.4.18 int AK_init_system_catalog ( )

Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function AK_register_system_tables() to register system tables.

Author

Miroslav Policki

Returns

EXIT_SUCCESS if the system catalog has been successfully initialized, EXIT_ERROR otherwise

5.2.4.19 int AK_init_system_tables_catalog (int *relation*, int *attribute*, int *index*, int *view*, int *sequence*, int *function*, int *function_arguments*, int *trigger*, int *trigger_conditions*, int *db*, int *db_obj*, int *user*, int *group*, int *user_group*, int *user_right*, int *group_right*, int *constraint*, int *constraintNull*, int *constraintCheck*, int *constraintUnique*, int *reference*)

Function initialises the sytem table catalog and writes the result in first (0) block in db_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained_with and AK_free_space attributes are initialized. Names of various database elements are written in block.

Author

Matija Novak

Parameters

relation	address of system table of relation in db_file
attribute	address of system table of attribute in db_file
index	address of system table of index in db_file
view	address of system table of view in db_file
sequence	address of system table of sequence in db_file
function	address of system table of function in db_file
function_arguments	address of system table of function_arguments in db_file
trigger	address of system table of trigger in db_file
trigger_conditions	address of system table of trigger_conditions in db_file
db	address of system table of db in db_file
db_obj	address of system table of db_obj in db_file
user	address of system table of user in db_file
group	address of system table of group in db_file
user_group	address of system table of users associated with groups in db_file
user_right	address of system table of user right in db_file
group_right	address of system table of group right in db_file
constraint	address of system table of constraint in db_file
constraintNull	address of system table of constraintNull in db_file
constraintCheck	system table address for check constraint
reference	address of system table of reference in db_file

Returns

EXIT_SUCCESS if initialization was succesful if not returns EXIT_ERROR

first header attribute of catalog_block

second attribute of catalog_block

initialize other elements of block (adress, type, chained_with, AK_free_space)

using as an address for the first AK_free space in block->data

merge catalog_heder with heders created before

insert data and tuple_dict in block

call function for writing the block on the first place in the file (ie. first block is on position zero)

5.2.4.20 void AK_insert_entry (AK_block * *block_address,* int *type,* void * *entry_data,* int *i*)

Function for inserting entry in tuple_dict and data of a block. Address, type and size of catalog_tuple_dict are set. Free space of block is also set.

Author

Matija Novak

Parameters

block_adress	adress of a block in which we want insert data
type	type of entry_data
entry_data	(char) data which is inserted, can be int but must first be converted to char
i	(int) adress in tuple_dict array (example block_address->tuple_dict[i])

Returns

No return value because it gets the address of an block like a function parameter and works directly with the orginal block

using strlen becuse sizeof(entry_data) is always 4 copy data into bloc->data on start position bloc->AK_free_space

address of entry data in block->data

calculate next AK_free space for the next entry data

```
sizeof(entry_data)+1);///(sizeof(int));
```

no need for "+strlen(entry_data)" while "+1" is like "new line"

type of entry data

size of entry data

copy tuple_dict to block->tuple_dict[i] must use & becouse tuple_dict[i] is value and catalog_tuple_dict adress

5.2.4.21 void AK_memset_int (void * *block*, int *value*, size_t *num*)

Function that sets the first num ints of a block of memory to the specified value.

Author

Miroslav Policki

Parameters

block	pointer to the block of memory to fill
value	int value to be set
num	number of ints in the block of memory to be set

Returns

No return value

5.2.4.22 int AK_new_extent (int *start_address*, int *old_size*, int *extent_type*, AK_header * *header*)

Function alocates new extent of blocks. If argument "old_size" is 0 than size of extent is INITIAL_EXTENT_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is

incremented.

Author

Nikola Bakoš, updated by Dino Laktašiæ (fixed header BUG), refurbished by dv

Parameters

start_address	address (block number) to start searching for sufficient space
old_size	size of previous extent in same segment (in blocks)
extent_type	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP
header	pointer to header that should be written to the new extent (all blocks)

Returns

address (block number) of new extent if successful, EXIT_ERROR otherwise

var - How much of space is required for extent

vars for loop [for]

if the old_size is 0 then the size of new extent is INITIAL_EXTENT_SIZE

if some blocks are not succesfully allocated, which means that the extend allocation has FAILED

5.2.4.23 int AK_new_segment (char * *name*, int *type*, AK_header * *header*)

Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL_EXTENT_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL_EXTENT_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK_free.

Author

Tomislav Fotak, refurbished by dv

Parameters

name	(character pointer) name of segment
type	segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
header	(header pointer) pointer to header that should be written to the new extent (all blocks)

Returns

EXIT_SUCCESS for success or EXIT_ERROR if some error occurs

start address for segment because we can not allocate segment in block 0

5.2.4.24 int AK_print_block (AK_block * *block,* int *num,* char * *gg,* FILE * *fpp*)

Function that dumps block.

Author

dv

Returns

nothing

5.2.4.25 AK_block* AK_read_block (int *address*)

Function that reads a block at a given address (block number less than db_file_size). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.

Author

Markus Schatten, updated dv and Domagoj Šitum (thread-safe enabled)

Parameters

address	block number (address)

Returns

pointer to block allocated in memory

5.2.4.26 void* AK_read_block_for_testing (void * *address*)

This function is only for testing. It has to be there, because pthread_create only accepts void* function_name (void *) function format. So AK_read_block is no-go for pthread_create.

Author

Domagoj Šitum

5.2.4.27 int AK_register_system_tables (int *relation,* int *attribute,* int *index,* int *view,* int *sequence,* int *function,* int *function_arguments,* int *trigger,* int *trigger_conditions,* int *db,* int *db_obj,* int *user,* int *group,* int *user_group,* int *user_right,* int *group_right,* int *constraint,* int *constraintNull,* int *constraintCheck,* int *constraintUnique,* int *reference*)

Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.

Author

Unknown

Parameters

relation	relation in database
attribute	attribute in databse
index	index in database
view	view in database
sequence	sequence in database
function	function in database
function_arguments	functional_arguments in databse
trigger	trigger in database
trigger_conditions	trigger conditions in databse
db	database
db_obj	database object
user	user in database
group	group in database
user_group	user associated with group in database
user_right	user right in database
group_right	group right in database
constraint	constraint in database
constraintNull	Null constraint in database
constraintCheck	Check constraint in database
reference	reference database

Returns

EXIT_SUCCESS

5.2.4.28 void AK_thread_safe_block_access_test ( )

This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.

Author

Domagoj Šitum

5.2.4.29 int AK_write_block (AK_block * *block*)

Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.

Author

Markus Schatten, updated by Domagoj Šitum (thread-safe enabled)

Parameters

block	poiner to block allocated in memory to write

Returns

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

5.2.4.30 void* AK_write_block_for_testing (void * *block*)

This function is only for testing. It has to be there, because pthread_create only accepts void* function_name (void *) function format. So AK_write_block is no-go for pthread_create.

Author

Domagoj Šitum

5.2.4.31 int fsize (FILE * *fp*)

Helper function to determine file size.

Returns

file size

5.2.5 Variable Documentation

5.2.5.1 AK_allocationbit

Global variable that holds allocation bit-vector.

Author

dv

5.2.5.2 db

Variable that defines the DB file file handle.

Author

Markus Schatten

5.2.5.3 db_file_size

Variable that defines the size of the DB file (in blocks)

Author

Markus Schatten

5.3 file/blobs.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include "../dm/dbman.h"
#include "../auxi/configuration.h"
#include "blobs.h"
```

Include dependency graph for blobs.c:

Functions

- AK_File_Metadata **AK_File_Metadata_malloc** ()
- char * AK_GUID ()

  Function for generating GUID.
- int AK_folder_exists (char *foldername)

  Function for checking if folder blobs already exists.
- int AK_mkdir (const char *path)

  Function for creating new folder.
- int **AK_copy** (const char *from, const char *to)
- char * AK_concat (char *s1, char *s2)

  Function for AK_concatinating 2 strings.
- char * **AK_clear_all_newline** (char *s)
- int AK_check_folder_blobs ()

  Function for checking if folder blobs exists.
- void AK_split_path_file (char **p, char **f, char *pf)

  Function for spliting path from filename.
- int **AK_write_metadata** (char *oid, AK_File_Metadata meta)
- AK_File_Metadata **AK_read_metadata** (char *oid)
- char * AK_lo_import (char *filepath)

  Function for importing large objects to database.
- int AK_lo_export (char *oid, char *filepath)

  Function for retrieving large objects.
- int AK_lo_unlink (char *oid)

  Function for deleting large objects.
- void AK_lo_test ()

  Tests.

5.3.1 Detailed Description

Provides functions for manipulations of binary large objects

5.3.2 Function Documentation

5.3.2.1 int AK_check_folder_blobs ( )

Function for checking if folder blobs exists.

Author

Samuel Picek

Returns

OID (object ID)

5.3.2.2 char* AK_concat (char * *s1*, char * *s2*)

Function for AK_concatinating 2 strings.

Author

Samuel Picek

Returns

returns new string

5.3.2.3 int AK_folder_exists (char * *foldername*)

Function for checking if folder blobs already exists.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.3.2.4 char* AK_GUID ( )

Function for generating GUID.

Author

Samuel Picek

Returns

returns globaly universal identifier based on kernel implementation

5.3.2.5 int AK_lo_export (char * *oid*, char * *filepath*)

Function for retrieving large objects.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.3.2.6 char* AK_lo_import (char * *filepath*)

Function for importing large objects to database.

Author

Samuel Picek

Returns

OID (object ID)

5.3.2.7 void AK_lo_test ( )

Tests.

Author

Samuel Picek

5.3.2.8 int AK_lo_unlink (char * *oid*)

Function for deleting large objects.

Author

Samuel Picek

Returns

OID (object ID)

5.3.2.9 int AK_mkdir (const char * *path*)

Function for creating new folder.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.3.2.10 void AK_split_path_file (char ** *p*, char ** *f*, char * *pf*)

Function for spliting path from filename.

Author

Samuel Picek

Returns

void

5.4 file/blobs.h File Reference

```
#include "table.h"
#include "fileio.h"
#include "id.h"
```

Include dependency graph for blobs.h: This graph shows which files directly or indirectly include this file:

Classes

- struct _file_metadata

Typedefs

- typedef struct _file_metadata **AK_Metadata**
- typedef struct _file_metadata * **AK_File_Metadata**

Functions

- AK_File_Metadata **AK_File_Metadata_malloc** ()
- int AK_mkdir (const char *path)

  Function for creating new folder.
- int **AK_copy** (const char *from, const char *to)
- char * AK_concat (char *s1, char *s2)

  Function for AK_concatinating 2 strings.
- char * **AK_clear_all_newline** (char *str)
- void AK_split_path_file (char **p, char **f, char *pf)

  Function for spliting path from filename.
- char * AK_GUID ()

  Function for generating GUID.
- int AK_folder_exists (char *foldername)

  Function for checking if folder blobs already exists.
- int AK_check_folder_blobs ()

  Function for checking if folder blobs exists.
- int **AK_write_metadata** (char *oid, AK_File_Metadata meta)
- AK_File_Metadata **AK_read_metadata** (char *oid)
- char * AK_lo_import (char *filepath)

  Function for importing large objects to database.
- int AK_lo_export (char *oid, char *filepath)

  Function for retrieving large objects.
- int AK_lo_unlink (char *oid)

  Function for deleting large objects.
- void AK_lo_test ()

  Tests.

5.4.1 Detailed Description

Provides data structures for manipulating blobs

5.4.2 Function Documentation

5.4.2.1 int AK_check_folder_blobs ( )

Function for checking if folder blobs exists.

Author

Samuel Picek

Returns

OID (object ID)

5.4.2.2 char* AK_concat (char * *s1*, char * *s2*)

Function for AK_concatinating 2 strings.

Author

Samuel Picek

Returns

returns new string

5.4.2.3 int AK_folder_exists (char * *foldername*)

Function for checking if folder blobs already exists.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.4.2.4 char* AK_GUID ( )

Function for generating GUID.

Author

Samuel Picek

Returns

returns globaly universal identifier based on kernel implementation

5.4.2.5 int AK_lo_export (char * *oid*, char * *filepath*)

Function for retrieving large objects.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.4.2.6 char* AK_lo_import (char * *filepath*)

Function for importing large objects to database.

Author

Samuel Picek

Returns

OID (object ID)

5.4.2.7 void AK_lo_test ( )

Tests.

Author

Samuel Picek

5.4.2.8 int AK_lo_unlink (char * *oid*)

Function for deleting large objects.

Author

Samuel Picek

Returns

OID (object ID)

5.4.2.9 int AK_mkdir (const char * *path*)

Function for creating new folder.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.4.2.10 void AK_split_path_file (char ** p, char ** f, char * pf)

Function for spliting path from filename.

Author

Samuel Picek

Returns

void

5.5 file/fileio.c File Reference

```
#include "fileio.h"
```

Include dependency graph for fileio.c:

Functions

- void Ak_Insert_New_Element_For_Update (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore, int newconstraint)

  Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.
- void Ak_Insert_New_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.
- int Ak_insert_row_to_block (struct list_node *row_root, AK_block *temp_block)

  Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last_tuple_id are put in temp_block.
- int Ak_insert_row (struct list_node *row_root)

  Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_DIRTY.
- void Ak_update_row_from_block (AK_block *temp_block, struct list_node *row_root)

  Function updates row from table in given block.
- void Ak_delete_row_from_block (AK_block *temp_block, struct list_node *row_root)

  Function deletes row from table in given block. Given list of elements is firstly back-upped.
- int Ak_delete_update_segment (struct list_node *row_root, int del)

  Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.
- int Ak_delete_row (struct list_node *row_root)

  Function deletes rows.
- void Ak_delete_row_by_id (int id, char *tableName)

  Function deletes row by id.
- int Ak_update_row (struct list_node *row_root)

  Function updates rows of some table.
- void **Ak_fileio_test** ()

5.5.1 Detailed Description

Provides functions for file input/output

5.5.2 Function Documentation

5.5.2.1 int Ak_delete_row (struct list_node * *row_root*)

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row EXIT_SUCCESS if success

5.5.2.2 void Ak_delete_row_by_id (int *id,* char * *tableName*)

Function deletes row by id.

Author

Dražen Bandić

Parameters

id	id of row
tableName	name of table to delete the row

5.5.2.3 void Ak_delete_row_from_block (AK_block * *temp_block,* struct list_node * *row_root*)

Function deletes row from table in given block. Given list of elements is firstly back-upped.

Author

Matija Novak, updated by Dino Laktašić, changed by Davorin Vukelic, updated by Mario Peroković

Parameters

temp_block	block to work with
row_list	list of elements which contain data for delete or update

Returns

No return value

5.5.2.4 int Ak_delete_update_segment (struct list_node * *row_root,* int *del*)

Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.

Author

Matija Novak, updated by Matija Šestak (function now uses caching)

Parameters

row_root	elements of one row
del	- DELETE or UPDATE

Returns

EXIT_SUCCESS if success

5.5.2.5 void Ak_Insert_New_Element (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore*)

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	is NEW_VALUE

Returns

No return value

5.5.2.6 void Ak_Insert_New_Element_For_Update (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore,* int *newconstraint*)

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

Returns

No return value

5.5.2.7 int Ak_insert_row (struct list_node * *row_root*)

Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_↩ DIRTY.

Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK_free, variable table initialized using memset)

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.5.2.8 int Ak_insert_row_to_block (struct list_node * *row_root,* AK_block * *temp_block*)

Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last_tuple_id are put in temp_block.

Author

Matija Novak, updated by Dino Laktašić

Parameters

row_root	list of elements to insert
temp_block	block in which we insert data

Returns

EXIT SUCCES if success

5.5.2.9 int Ak_update_row (struct list_node * *row_root*)

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.5.2.10 void Ak_update_row_from_block (AK_block * *temp_block,* struct list_node * *row_root*)

Function updates row from table in given block.

Author

Matija Novak, updated by Dino Laktašić, updated by Mario Peroković - separated from deletion

Parameters

temp_block	block to work with
row_list	list of elements which contain data for delete or update

Returns

No return value

5.6 file/fileio.h File Reference

```
#include "../sql/cs/reference.h"
#include "../mm/memoman.h"
#include "../rec/recovery.h"
#include "../rec/redo_log.h"
#include "files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for fileio.h: This graph shows which files directly or indirectly include this file:

Functions

- void Ak_Insert_New_Element_For_Update (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore, int newconstraint)

  Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.
- void Ak_Insert_New_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.
- int Ak_insert_row_to_block (struct list_node *row_root, AK_block *temp_block)

  Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last_tuple_id are put in temp_block.
- int Ak_insert_row (struct list_node *row_root)

  Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_DIRTY.
- void Ak_update_row_from_block (AK_block *temp_block, struct list_node *row_root)

  Function updates row from table in given block.
- void Ak_delete_row_from_block (AK_block *temp_block, struct list_node *row_root)

  Function deletes row from table in given block. Given list of elements is firstly back-upped.
- int Ak_delete_update_segment (struct list_node *row_root, int del)

  Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.
- int Ak_delete_row (struct list_node *row_root)

  Function deletes rows.
- int Ak_update_row (struct list_node *row_root)

  Function updates rows of some table.
- void **Ak_fileio_test** ()
- void Ak_delete_row_by_id (int id, char *tableName)

  Function deletes row by id.

5.6.1 Detailed Description

Header file provides data structures for file input/output

5.6.2 Function Documentation

5.6.2.1 int Ak_delete_row (struct list_node * *row_root*)

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row EXIT_SUCCESS if success

5.6.2.2 void Ak_delete_row_by_id (int *id,* char * *tableName*)

Function deletes row by id.

Author

Dražen Bandić

Parameters

id	id of row
tableName	name of table to delete the row

5.6.2.3 void Ak_delete_row_from_block (AK_block * *temp_block,* struct list_node * *row_root*)

Function deletes row from table in given block. Given list of elements is firstly back-upped.

Author

Matija Novak, updated by Dino Laktašić, changed by Davorin Vukelic, updated by Mario Peroković

Parameters

temp_block	block to work with
row_list	list of elements which contain data for delete or update

Returns

No return value

5.6.2.4 int Ak_delete_update_segment (struct list_node * *row_root,* int *del*)

Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.

Author

Matija Novak, updated by Matija Šestak (function now uses caching)

Parameters

row_root	elements of one row
del	- DELETE or UPDATE

Returns

EXIT_SUCCESS if success

5.6.2.5 void Ak_Insert_New_Element (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore*)

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	is NEW_VALUE

Returns

No return value

5.6.2.6 void Ak_Insert_New_Element_For_Update (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore,* int *newconstraint*)

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

Returns

No return value

5.6.2.7 int Ak_insert_row (struct list_node * *row_root*)

Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_↩ DIRTY.

Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK_free, variable table initialized using memset)

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.6.2.8 int Ak_insert_row_to_block (struct list_node * *row_root,* AK_block * *temp_block*)

Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last_tuple_id are put in temp_block.

Author

Matija Novak, updated by Dino Laktašić

Parameters

row_root	list of elements to insert
temp_block	block in which we insert data

Returns

EXIT SUCCES if success

5.6.2.9 int Ak_update_row (struct list_node * *row_root*)

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.6.2.10 void Ak_update_row_from_block (AK_block * *temp_block*, struct list_node * *row_root*)

Function updates row from table in given block.

Author

Matija Novak, updated by Dino Laktašić, updated by Mario Peroković - separated from deletion

Parameters

temp_block	block to work with
row_list	list of elements which contain data for delete or update

Returns

No return value

5.7 file/files.c File Reference

```
#include "files.h"
#include <pthread.h>
```

Include dependency graph for files.c:

Functions

- int AK_initialize_new_segment (char *name, int type, AK_header *header)

  Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- int AK_initialize_new_index_segment (char *name, char *table_id, int attr_id, AK_header *header)

  Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- void Ak_files_test ()

  Test function.

Variables

- pthread_mutex_t **fileMut** = PTHREAD_MUTEX_INITIALIZER

5.7.1 Detailed Description

Header file provides functions for file management

5.7.2 Function Documentation

5.7.2.1 void Ak_files_test ( )

Test function.

Author

Unknown

Returns

No return value

5.7.2.2 int AK_initialize_new_index_segment (char * *name,* char * *table_id,* int *attr_id,* AK_header * *header*)

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching), reused by Lovro Predovan

Parameters

name	segment name
type	segment type
header	pointer to header that should be written to the new extent (all blocks)

Returns

start address of new segment

5.7.2.3 int AK_initialize_new_segment (char * *name,* int *type,* AK_header * *header*)

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

Parameters

name	segment name
type	segment type
header	pointer to header that should be written to the new extent (all blocks)

Returns

start address of new segment

5.8 file/files.h File Reference

```
#include "id.h"
#include "../auxi/mempro.h"
```

Include dependency graph for files.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_initialize_new_segment (char *name, int type, AK_header *header)

  Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- int AK_initialize_new_index_segment (char *name, char *table_id, int attr_id, AK_header *header)

  Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- void Ak_files_test ()

  Test function.

5.8.1 Detailed Description

Header file that provides data structures for file management

5.8.2 Function Documentation

5.8.2.1 void Ak_files_test ( )

Test function.

Author

Unknown

Returns

No return value

5.8.2.2 int AK_initialize_new_index_segment (char * *name*, char * *table_id*, int *attr_id*, AK_header * *header*)

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching), reused by Lovro Predovan

Parameters

name	segment name
type	segment type
header	pointer to header that should be written to the new extent (all blocks)

Returns

start address of new segment

5.8.2.3 int AK_initialize_new_segment (char * *name*, int *type*, AK_header * *header*)

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

Parameters

name	segment name
type	segment type
header	pointer to header that should be written to the new extent (all blocks)

Returns

start address of new segment

5.9 file/filesearch.c File Reference

`#include "filesearch.h"`
Include dependency graph for filesearch.c:

Functions

- search_result AK_search_unsorted (char *szRelation, search_params *aspParams, int iNum_search_↩ params)

  Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_R↩ ANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.
- void AK_deallocate_search_result (search_result srResult)

  Function deallocates memory used by search result returned by AK_search_unsorted.
- void Ak_filesearch_test ()

  Function for testing file search.

5.9.1 Detailed Description

Provides functions for file searching

5.9.2 Function Documentation

5.9.2.1 void AK_deallocate_search_result (search_result *srResult*)

Function deallocates memory used by search result returned by AK_search_unsorted.

Author

Miroslav Policki

Parameters

srResult	search result

Returns

No return value

5.9.2.2 void Ak_filesearch_test ( )

Function for testing file search.

Author

Miroslav Policki

Returns

No return value

5.9.2.3 search_result AK_search_unsorted (char * *szRelation,* search_params * *aspParams,* int *iNum_search_params*)

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_RANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

Author

Miroslav Policki

Parameters

szRelation	relation name
aspParams	array of search parameters
iNum_search_params	number of search parameters

Returns

search_result structure defined in filesearch.h. Use AK_deallocate_search_result to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

5.10 file/filesearch.h File Reference

```
#include "../mm/memoman.h"
#include "files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for filesearch.h: This graph shows which files directly or indirectly include this file:

Classes

- struct search_params

  Structure that contains attribute name, lower and upper data value, special(NULL or ∗) which is input for AK_↩ equisearch_unsorted and AK_rangesearch_unsorted.
- struct search_result

  Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted.

Macros

- #define **SEARCH_NULL** 0
- #define **SEARCH_ALL** 1
- #define **SEARCH_PARTICULAR** 2
- #define **SEARCH_RANGE** 3

Functions

- search_result AK_search_unsorted (char *szRelation, search_params *aspParams, int iNum_search_↩ params)

  Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_R↩ ANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.
- void AK_deallocate_search_result (search_result srResult)

  Function deallocates memory used by search result returned by AK_search_unsorted.
- void Ak_filesearch_test ()

  Function for testing file search.

5.10.1 Detailed Description

Header file provides data structures for file searching

5.10.2 Function Documentation

5.10.2.1 void AK_deallocate_search_result (search_result *srResult*)

Function deallocates memory used by search result returned by AK_search_unsorted.

Author

Miroslav Policki

Parameters

srResult	search result

Returns

No return value

5.10.2.2 void Ak_filesearch_test ( )

Function for testing file search.

Author

Miroslav Policki

Returns

No return value

5.10.2.3 search_result AK_search_unsorted (char * *szRelation,* search_params * *aspParams,* int *iNum_search_params*)

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_RANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

Author

Miroslav Policki

Parameters

szRelation	relation name
aspParams	array of search parameters
iNum_search_params	number of search parameters

Returns

search_result structure defined in filesearch.h. Use AK_deallocate_search_result to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

5.11 file/filesort.h File Reference

```
#include "../mm/memoman.h"
#include "table.h"
#include "files.h"
#include "fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for filesort.h:

Macros

- #define DATA_ROW_SIZE 200

  Constatnt declaring size of data to be compared.
- #define DATA_TUPLE_SIZE 500

  Costant declaring size of data to be copied.

Functions

- int Ak_get_total_headers (AK_block *iBlock)

  Function returns total number of headers in the block.
- int Ak_get_header_number (AK_block *iBlock, char *attribute_name)

  Function returns number of header in the block which to sort.
- int Ak_get_num_of_tuples (AK_block *iBlock)

  Function returns tuples number in block.
- void **AK_sort_segment** (char *table_name, char *attr)
- void **Ak_reset_block** (AK_block *block)
- void AK_block_sort (AK_block *iBlock, char *atr_name)

  Function sorts the given block.
- void **Ak_filesort_test** ()

5.11.1 Detailed Description

Header filr provides data structures for file sorting

5.11.2 Function Documentation

5.11.2.1 void AK_block_sort (AK_block * *iBlock,* char * *atr_name*)

Function sorts the given block.

Author

Bakoš Nikola

Version

v1.0

Parameters

iBlock	block to be sorted

Returns

No return value

5.11.2.2 int Ak_get_header_number (AK_block * *iBlock,* char * *attribute_name*)

Function returns number of header in the block which to sort.

Author

Unknown

Returns

number of attribute in header (0 - MAX_ATTRIBUTES). USE in tuple_dict[num]...

5.11.2.3 int Ak_get_num_of_tuples (AK_block * *iBlock*)

Function returns tuples number in block.

Author

Unknown

Returns

tuples number in block

5.11.2.4 int Ak_get_total_headers (AK_block * *iBlock*)

Function returns total number of headers in the block.

Author

Unknown

Returns

number of attribute in header (0 - MAX_ATTRIBUTES). USE in tuple_dict[num]...

5.12 file/id.c File Reference

```
#include "id.h"
```

Include dependency graph for id.c:

Functions

- int AK_get_id ()

  Function for getting unique ID for any object, stored in sequence.
- char AK_get_table_id (char *tableName)

  Function for getting unique ID for any object, stored in sequence based on table name.
- void Ak_id_test ()

  Function for testing getting ID's.

5.12.1 Detailed Description

Provides functions for creating id of objects

5.12.2 Function Documentation

5.12.2.1 int AK_get_id ( )

Function for getting unique ID for any object, stored in sequence.

Author

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses Ak_update_row, updated by Nenad Makar

Returns

objectID

5.12.2.2 char AK_get_table_id (char * *tableName*)

Function for getting unique ID for any object, stored in sequence based on table name.

Author

Lovro Predovan

Returns

objectID in string(char) format

5.12.2.3 void Ak_id_test ( )

Function for testing getting ID's.

Author

Mislav Čakarić, updated by Nenad Makar

Returns

No retun value

5.13 file/id.h File Reference

```
#include "table.h"
#include "fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for id.h: This graph shows which files directly or indirectly include this file:

Macros

- #define ID_START_VALUE 100

  Constant declaring start value of id.

Functions

- int AK_get_id ()

  Function for getting unique ID for any object, stored in sequence.
- void Ak_id_test ()

  Function for testing getting ID's.

5.13.1 Detailed Description

Provides functions, data structures and constants for creating id of objects

5.13.2 Function Documentation

5.13.2.1 int AK_get_id ( )

Function for getting unique ID for any object, stored in sequence.

Author

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses Ak_update_row, updated by Nenad Makar

Returns

objectID

5.13.2.2 void Ak_id_test ( )

Function for testing getting ID's.

Author

Mislav Čakarić, updated by Nenad Makar

Returns

No retun value

5.14 file/idx/bitmap.c File Reference

```
#include "bitmap.h"
#include "../../auxi/iniparser.h"
```

Include dependency graph for bitmap.c:

Functions

- int Ak_If_ExistOp (struct list_node *L, char *ele)

  Function examines whether list L contains operator ele.
- void AK_create_Index_Table (char *tblName, struct list_node *attributes)

  Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.
- void Ak_create_Index (char *tblName, char *tblNameIndex, char *attributeName, int positionTbl, int numAtributes, AK_header *headerIndex)

  Function that loads index table with value of particulary atribute.
- list_ad * Ak_get_Attribute (char *indexName, char *attribute)

  Function gets adresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add_root.
- void Ak_print_Att_Test (list_ad *list)

  Function for printing list of adresses.
- list_ad * AK_get_Attribute (char *tableName, char *attributeName, char *attributeValue)

  Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.
- void AK_update (int addBlock, int addTd, char *tableName, char *attributeName, char *attributeValue, char *newAttributeValue)

  Function for updating the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.
- int Ak_write_block (AK_block *block)

  Function for writing new value in block when index is updated.
- void AK_add_to_bitmap_index (char *tableName, char *attributeName)

  Function for updating the index,function deletes and recrates index values again if different number of params is detected.
- void Ak_print_Header_Test (char *tblName)

  Function that tests printing header of table.
- void AK_delete_bitmap_index (char *indexName)

  Function that deletes bitmap index based on name of index.
- void Ak_bitmap_test ()

  Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.

5.14.1 Detailed Description

Provides functions for bitmap indexes

5.14.2 Function Documentation

5.14.2.1 void AK_add_to_bitmap_index (char * *tableName,* char * *attributeName*)

Function for updating the index,function deletes and recrates index values again if different number of params is detected.

Author

Lovro Predovan

Parameters

tableName	name of table
attributeName	name of attribute
newAttributeValue	new value of updated attribute

Returns

No return value

5.14.2.2 void Ak_bitmap_test ( )

Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.

Author

Saša Vukšić updated by Lovro Predovan

Returns

No return value

5.14.2.3 void Ak_create_Index (char * *tblName,* char * *tblNameIndex,* char * *attributeName,* int *positionTbl,* int *numAtributes,* AK_header * *headerIndex*)

Function that loads index table with value of particulary atribute.

Author

Saša Vukšić, Lovro Predovan

Parameters

tblName	source table
tblNameIndex	new name of index table
attributeName	attribute on which we make index
positionTbl	position of attribute in header of table
numAtributes	number of attributes in table
headerIndex	header of index table

Returns

No return value

5.14.2.4 void AK_create_Index_Table (char * *tblName,* struct list_node * *attributes*)

Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.

Author

Saša Vukšić, Lovro Predovan

Parameters

tblName	name of table
attributes	list of attributes on which we will create indexes

Returns

No return value

5.14.2.5 void AK_delete_bitmap_index (char * *indexName*)

Function that deletes bitmap index based on name of index.

Author

Lovro Predovan

Parameters

Bitmap	index table name

Returns

No return value

5.14.2.6 list_ad* Ak_get_Attribute (char * *indexName,* char * *attribute*)

Function gets adresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add_root.

Author

Saša Vukšić, Lovro Predovan

Parameters

indexName	name of index
attribute	name of attribute

Returns

list of adresses

5.14.2.7 list_ad* AK_get_Attribute (char * *tableName,* char * *attributeName,* char * *attributeValue*)

Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.

Author

Saša Vukšić

Parameters

tableName	name of table
attributeValue	value of attribute

Returns

list of adresses

5.14.2.8 int Ak_If_ExistOp (struct list_node * *L,* char * *ele*)

Function examines whether list L contains operator ele.

Author

Saša Vukšić

Parameters

L	list of elements
ele	operator to be found in list

Returns

1 if operator ele is found in list, otherwise 0

5.14.2.9 void Ak_print_Att_Test (list_ad * *list*)

Function for printing list of adresses.

Author

Saša Vukšić, Lovro Predovan

Parameters

list	list of adresses

Returns

No return value

5.14.2.10 void Ak_print_Header_Test (char * *tblName*)

Function that tests printing header of table.

Author

Saša Vukšić

Parameters

tblName	name of table who's header we are printing

Returns

No return value

5.14.2.11 void AK_update (int *addBlock,* int *addTd,* char * *tableName,* char * *attributeName,* char * *attributeValue,* char * *newAttributeValue*)

Function for updating the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

Author

Saša Vukšić

Parameters

addBlock	adress of block
addTD	adress of tuple dict
tableName	name of table
attributeName	name of attribute
attributeValue	value of atribute
newAttributeValue	new value of updated attribute

Returns

No return value

5.14.2.12 int Ak_write_block (AK_block * *block*)

Function for writing new value in block when index is updated.

Author

Saša Vukšić

Parameters

block	block to write on

Returns

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

5.15 file/idx/bitmap.h File Reference

```
#include "../../mm/memoman.h"
#include "index.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../file/files.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for bitmap.h: This graph shows which files directly or indirectly include this file:

Functions

- int Ak_If_ExistOp (struct list_node *L, char *ele)

  Function examines whether list L contains operator ele.
- void AK_create_Index_Table (char *tblName, struct list_node *attributes)

  Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.
- void Ak_print_Header_Test (char *tblName)

  Function that tests printing header of table.
- void Ak_create_Index (char *tblName, char *tblNameIndex, char *attributeName, int positionTbl, int numAtributes, AK_header *headerIndex)

  Function that loads index table with value of particulary atribute.
- list_ad * Ak_get_Attribute (char *indexName, char *attribute)

  Function gets adresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add_root.
- void **Ak_create_List_Address_Test** ()
- void Ak_print_Att_Test (list_ad *list)

  Function for printing list of adresses.

- list_ad * AK_get_Attribute (char *tableName, char *attributeName, char *attributeValue)

  Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.
- void AK_update (int addBlock, int addTd, char *tableName, char *attributeName, char *attributeValue, char *newAttributeValue)

  Function for updating the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.
- int Ak_write_block (AK_block *block)

  Function for writing new value in block when index is updated.
- void Ak_bitmap_test ()

  Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.
- void AK_delete_bitmap_index (char *indexName)

  Function that deletes bitmap index based on name of index.
- void AK_add_to_bitmap_index (char *tableName, char *attributeName)

  Function for updating the index,function deletes and recrates index values again if different number of params is detected.

5.15.1 Detailed Description

Header file that provides data structures for bitmap index

5.15.2 Function Documentation

5.15.2.1 void AK_add_to_bitmap_index (char * *tableName*, char * *attributeName*)

Function for updating the index,function deletes and recrates index values again if different number of params is detected.

Author

Lovro Predovan

Parameters

tableName	name of table
attributeName	name of attribute
newAttributeValue	new value of updated attribute

Returns

No return value

5.15.2.2 void Ak_bitmap_test ( )

Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.

Author

Saša Vukšić updated by Lovro Predovan

Returns

No return value

5.15.2.3 void Ak_create_Index (char * *tblName,* char * *tblNameIndex,* char * *attributeName,* int *positionTbl,* int *numAtributes,* AK_header * *headerIndex*)

Function that loads index table with value of particulary atribute.

Author

Saša Vukšić, Lovro Predovan

Parameters

tblName	source table
tblNameIndex	new name of index table
attributeName	attribute on which we make index
positionTbl	position of attribute in header of table
numAtributes	number of attributes in table
headerIndex	header of index table

Returns

No return value

5.15.2.4 void AK_create_Index_Table (char * *tblName,* struct list_node * *attributes*)

Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.

Author

Saša Vukšić, Lovro Predovan

Parameters

tblName	name of table
attributes	list of attributes on which we will create indexes

Returns

No return value

5.15.2.5 void AK_delete_bitmap_index (char * *indexName*)

Function that deletes bitmap index based on name of index.

Author

Lovro Predovan

Parameters

Bitmap	index table name

Returns

No return value

5.15.2.6 list_ad* Ak_get_Attribute (char * *indexName,* char * *attribute*)

Function gets adresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add_root.

Author

Saša Vukšić, Lovro Predovan

Parameters

indexName	name of index
attribute	name of attribute

Returns

list of adresses

5.15.2.7 list_ad* AK_get_Attribute (char * *tableName,* char * *attributeName,* char * *attributeValue*)

Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.

Author

Saša Vukšić

Parameters

tableName	name of table
attributeValue	value of attribute

Returns

list of adresses

5.15.2.8 int Ak_If_ExistOp (struct list_node * *L*, char * *ele*)

Function examines whether list L contains operator ele.

Author

Saša Vukšić

Parameters

L	list of elements
ele	operator to be found in list

Returns

1 if operator ele is found in list, otherwise 0

5.15.2.9 void Ak_print_Att_Test (list_ad * *list*)

Function for printing list of adresses.

Author

Saša Vukšić, Lovro Predovan

Parameters

list	list of adresses

Returns

No return value

5.15.2.10 void Ak_print_Header_Test (char * *tblName*)

Function that tests printing header of table.

Author

Saša Vukšić

Parameters

tblName	name of table who's header we are printing

Returns

No return value

5.15.2.11 void AK_update (int *addBlock*, int *addTd*, char * *tableName*, char * *attributeName*, char * *attributeValue*, char * *newAttributeValue*)

Function for updating the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

Author

Saša Vukšić

Parameters

addBlock	adress of block
addTD	adress of tuple dict
tableName	name of table
attributeName	name of attribute
attributeValue	value of atribute
newAttributeValue	new value of updated attribute

Returns

No return value

5.15.2.12 int Ak_write_block (AK_block * *block*)

Function for writing new value in block when index is updated.

Author

Saša Vukšić

Parameters

block	block to write on

Returns

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

5.16 file/idx/btree.c File Reference

`#include "btree.h"`
Include dependency graph for btree.c:

Functions

- int AK_btree_create (char *tblName, struct list_node *attributes, char *indexName)
  Function for creating new btree index on integer attribute in table.
- int **AK_btree_delete** (char *indexName)
- void AK_btree_search_delete (char *indexName, int *searchValue, int *endRange, int *toDo)
  Function for searching or deleting a value in btree index.
- int **AK_btree_insert** (char *indexName, int *insertValue, int *insertTd, int *insertBlock)
- void **Ak_btree_test** ()

5.16.1 Detailed Description

Header file that provides functions for BTree indices

5.16.2 Function Documentation

5.16.2.1 int AK_btree_create (char * *tblName*, struct list_node * *attributes*, char * *indexName*)

Function for creating new btree index on integer attribute in table.

Author

Anđelko Spevec

Parameters

tblName	- name of the table on which we are creating index
attributes	- attribute on which we are creating index
indexName	- name of the index

5.16.2.2 void AK_btree_search_delete (char * *indexName*, int * *searchValue*, int * *endRange*, int * *toDo*)

Function for searching or deleting a value in btree index.

Author

Anđelko Spevec

Parameters

indexName	- name of the index
searchValue	- value that we are searching in the index
endRange	- if 0 search is for 0 value, else searching in range
toDo	- if 0 we just search else we delete the element if we find it

5.17 file/idx/btree.h File Reference

```
#include "index.h"
#include "../../file/table.h"
#include "../../auxi/constants.h"
#include "../../auxi/configuration.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for btree.h: This graph shows which files directly or indirectly include this file:

Classes

- struct btree_node
- struct root_info

Macros

- #define **B** 3
- #define **ORDER** 6
- #define **LEAF** 0
- #define **NODE** 1

Functions

- int AK_btree_create (char ∗tblName, struct list_node ∗attributes, char ∗indexName)

  Function for creating new btree index on integer attribute in table.
- int **AK_btree_delete** (char ∗indexName)
- void AK_btree_search_delete (char ∗indexName, int ∗searchValue, int ∗endRange, int ∗toDo)

  Function for searching or deleting a value in btree index.
- int **AK_btree_insert** (char ∗indexName, int ∗insertValue, int ∗insertTd, int ∗insertBlock)
- void **Ak_btree_test** ()

5.17.1 Detailed Description

Header file that provides data strucures for BTree indices

5.17.2 Function Documentation

5.17.2.1 int AK_btree_create (char ∗ *tblName,* struct list_node ∗ *attributes,* char ∗ *indexName*)

Function for creating new btree index on integer attribute in table.

Author

Anđelko Spevec

Parameters

tblName	- name of the table on which we are creating index
attributes	- attribute on which we are creating index
indexName	- name of the index

5.17.2.2 void AK_btree_search_delete (char * *indexName*, int * *searchValue*, int * *endRange*, int * *toDo*)

Function for searching or deleting a value in btree index.

Author

Anđelko Spevec

Parameters

indexName	- name of the index
searchValue	- value that we are searching in the index
endRange	- if 0 search is for 0 value, else searching in range
toDo	- if 0 we just search else we delete the element if we find it

5.18 file/idx/hash.c File Reference

```
#include "hash.h"
```

Include dependency graph for hash.c:

Functions

- int AK_elem_hash_value (struct list_node *elem)

  Function for computing a hash value from varchar or integer.
- struct_add * Ak_insert_bucket_to_block (char *indexName, char *data, int type)

  Function for inserting bucket to block.
- void Ak_update_bucket_in_block (struct_add *add, char *data)

  Function for update bucket in block.
- void AK_change_hash_info (char *indexName, int modulo, int main_bucket_num, int hash_bucket_num)

  Function for changing info of hash index.
- hash_info * AK_get_hash_info (char *indexName)

  Function for fetching info for hash index.
- struct_add * Ak_get_nth_main_bucket_add (char *indexName, int n)

  Function for fetching nth main bucket.
- void AK_insert_in_hash_index (char *indexName, int hashValue, struct_add *add)

  Function for inserting record in hash bucket.
- struct_add * AK_find_delete_in_hash_index (char *indexName, struct list_node *values, int delete)

  Function for fetching or deleting record from hash index.
- struct_add * AK_find_in_hash_index (char *indexName, struct list_node *values)

Function for fetching record from hash index.

- void AK_delete_in_hash_index (char ∗indexName, struct list_node ∗values)

  Function for deleting record from hash index.
- int AK_create_hash_index (char ∗tblName, struct list_node ∗attributes, char ∗indexName)

  Function for creating hash index.
- void **AK_delete_hash_index** (char ∗indexName)
- void Ak_hash_test ()

  Function for testing hash index.

5.18.1 Detailed Description

Provides functions for Hash indices

5.18.2 Function Documentation

5.18.2.1 void AK_change_hash_info (char ∗ *indexName,* int *modulo,* int *main_bucket_num,* int *hash_bucket_num*)

Function for changing info of hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
modulo	value for modulo hash function
main_bucket_num	number of main buckets
hash_bucket_num	number of hash buckets

Returns

No return value

5.18.2.2 int AK_create_hash_index (char ∗ *tblName,* struct list_node ∗ *attributes,* char ∗ *indexName*)

Function for creating hash index.

Author

Mislav Čakarić

Parameters

tblName	name of table for which the index is being created
indexName	name of index
attributes	list of attributes over which the index is being created

Returns

success or error

5.18.2.3 void AK_delete_in_hash_index (char * *indexName,* struct list_node * *values*)

Function for deleting record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index

Returns

No return value

5.18.2.4 int AK_elem_hash_value (struct list_node * *elem*)

Function for computing a hash value from varchar or integer.

Author

Mislav Čakarić

Parameters

elem	element of row for wich value is to be computed

Returns

hash value

5.18.2.5 struct_add* AK_find_delete_in_hash_index (char * *indexName,* struct list_node * *values,* int *delete*)

Function for fetching or deleting record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index
delete	if delete is 0 then record is only read otherwise it's deleted from hash index

Returns

address structure with data where the record is in table

5.18.2.6 struct_add* AK_find_in_hash_index (char * *indexName,* struct list_node * *values*)

Function for fetching record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index

Returns

address structure with data where the record is in table

5.18.2.7 hash_info* AK_get_hash_info (char * *indexName*)

Function for fetching info for hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index

Returns

info bucket with info data for hash index

5.18.2.8 struct_add* Ak_get_nth_main_bucket_add (char * *indexName,* int *n*)

Function for fetching nth main bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
n	number of main bucket

Returns

address structure with data where the bucket is stored

5.18.2.9 void Ak_hash_test ( )

Function for testing hash index.

Author

Mislav Čakarić

Returns

No return value

5.18.2.10 struct_add* Ak_insert_bucket_to_block (char * *indexName,* char * *data,* int *type*)

Function for inserting bucket to block.

Author

Mislav Čakarić

Parameters

indexName	name of index
data	content of bucket stored in char array
type	type of bucket (MAIN_BUCKET or HASH_BUCKET)

Returns

address structure with data where the bucket is stored

5.18.2.11 void AK_insert_in_hash_index (char * *indexName,* int *hashValue,* struct_add * *add*)

Function for inserting record in hash bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
hashValue	hash value of record that is being inserted
add	address structure with data where the hash bucket is stored

Returns

No return value

5.18.2.12 void Ak_update_bucket_in_block (struct_add * add, char * data)

Function for update bucket in block.

Author

Mislav Čakarić

Parameters

add	address of where the bucket is stored
data	content of bucket stored in char array

Returns

No return value

5.19 file/idx/hash.h File Reference

```
#include "index.h"
#include "../../file/table.h"
#include "../../auxi/constants.h"
#include "../../auxi/configuration.h"
#include "../files.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for hash.h: This graph shows which files directly or indirectly include this file:

Classes

- struct hash_info

  Structure for defining a hash info element.
- struct bucket_elem

  Structure for defining a single bucket element.
- struct main_bucket

  Structure for defining main bucket for table hashing.
- struct hash_bucket

  Structure for hash bucket for table hashing.

Functions

- int AK_elem_hash_value (struct list_node *elem)

  Function for computing a hash value from varchar or integer.
- struct_add * Ak_insert_bucket_to_block (char *indexName, char *data, int type)

  Function for inserting bucket to block.
- void Ak_update_bucket_in_block (struct_add *add, char *data)

  Function for update bucket in block.
- void AK_change_hash_info (char *indexName, int modulo, int main_bucket_num, int hash_bucket_num)

  Function for changing info of hash index.
- hash_info * AK_get_hash_info (char *indexName)

  Function for fetching info for hash index.
- struct_add * Ak_get_nth_main_bucket_add (char *indexName, int n)

  Function for fetching nth main bucket.
- void AK_insert_in_hash_index (char *indexName, int hashValue, struct_add *add)

  Function for inserting record in hash bucket.
- struct_add * AK_find_delete_in_hash_index (char *indexName, struct list_node *values, int delete)

  Function for fetching or deleting record from hash index.
- struct_add * AK_find_in_hash_index (char *indexName, struct list_node *values)

  Function for fetching record from hash index.
- void AK_delete_in_hash_index (char *indexName, struct list_node *values)

  Function for deleting record from hash index.
- int AK_create_hash_index (char *tblName, struct list_node *attributes, char *indexName)

  Function for creating hash index.
- void **AK_delete_hash_index** (char *indexName)
- void Ak_hash_test ()

  Function for testing hash index.

5.19.1 Detailed Description

Header file that provides data structures for Hash indices

5.19.2 Function Documentation

5.19.2.1 void AK_change_hash_info (char * *indexName*, int *modulo*, int *main_bucket_num*, int *hash_bucket_num*)

Function for changing info of hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
modulo	value for modulo hash function
main_bucket_num	number of main buckets
hash_bucket_num	number of hash buckets

Returns

No return value

5.19.2.2 int AK_create_hash_index (char * *tblName,* struct list_node * *attributes,* char * *indexName*)

Function for creating hash index.

Author

Mislav Čakarić

Parameters

tblName	name of table for which the index is being created
indexName	name of index
attributes	list of attributes over which the index is being created

Returns

success or error

5.19.2.3 void AK_delete_in_hash_index (char * *indexName,* struct list_node * *values*)

Function for deleting record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index

Returns

No return value

5.19.2.4 int AK_elem_hash_value (struct list_node * *elem*)

Function for computing a hash value from varchar or integer.

Author

Mislav Čakarić

Parameters

elem	element of row for wich value is to be computed

Returns

hash value

5.19.2.5 struct_add* AK_find_delete_in_hash_index (char * *indexName,* struct list_node * *values,* int *delete*)

Function for fetching or deleting record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index
delete	if delete is 0 then record is only read otherwise it's deleted from hash index

Returns

address structure with data where the record is in table

5.19.2.6 struct_add* AK_find_in_hash_index (char * *indexName,* struct list_node * *values*)

Function for fetching record from hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index
values	list of values (one row) to search in hash index

Returns

address structure with data where the record is in table

5.19.2.7 hash_info* AK_get_hash_info (char * *indexName*)

Function for fetching info for hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index

Returns

info bucket with info data for hash index

5.19.2.8 struct_add* Ak_get_nth_main_bucket_add (char * *indexName,* int *n*)

Function for fetching nth main bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
n	number of main bucket

Returns

address structure with data where the bucket is stored

5.19.2.9 void Ak_hash_test ( )

Function for testing hash index.

Author

Mislav Čakarić

Returns

No return value

5.19.2.10 struct_add* Ak_insert_bucket_to_block (char * *indexName,* char * *data,* int *type*)

Function for inserting bucket to block.

Author

Mislav Čakarić

Parameters

indexName	name of index
data	content of bucket stored in char array
type	type of bucket (MAIN_BUCKET or HASH_BUCKET)

Returns

address structure with data where the bucket is stored

5.19.2.11 void AK_insert_in_hash_index (char * *indexName*, int *hashValue*, struct_add * *add*)

Function for inserting record in hash bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
hashValue	hash value of record that is being inserted
add	address structure with data where the hash bucket is stored

Returns

No return value

5.19.2.12 void Ak_update_bucket_in_block (struct_add * *add*, char * *data*)

Function for update bucket in block.

Author

Mislav Čakarić

Parameters

add	address of where the bucket is stored
data	content of bucket stored in char array

Returns

No return value

5.20 file/idx/index.c File Reference

```
#include "index.h"
#include <stdlib.h>
#include "../../auxi/mempro.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../file/files.h"
```

Include dependency graph for index.c:

Functions

- void Ak_InitializelistAd (list_ad *L)

  Function for initalizing linked list.
- element_ad Ak_Get_First_elementAd (list_ad *L)

  Function for finding first node of linked list.
- element_ad Ak_Get_Last_elementAd (list_ad *L)

  Function for finding last node of linked list.
- element_ad Ak_Get_Next_elementAd (element_ad Currentelement_op)

  Function for finding the next node of a node in linked list.
- element_ad Ak_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  Function for finding the previous node of a node in linked list.
- int Ak_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad *L)

  Function for finding the position of a node in linked list.
- void Ak_Delete_elementAd (element_ad Deletedelement_op, list_ad *L)

  Function for deleting a node in linked list.
- void Ak_Delete_All_elementsAd (list_ad *L)

  Function for deleting all nodes in linked list.
- void Ak_Insert_NewelementAd (int addBlock, int indexTd, char *attName, element_ad elementBefore)

  Function for inserting a new element into linked list.
- int AK_num_index_attr (char *indexTblName)

  Function for getting number of elements in index table.
- int AK_get_index_num_records (char *indexTblName)

  Determine number of rows in the table.
- struct list_node * AK_get_index_tuple (int row, int column, char *indexTblName)

  Function that gets value in some row and column.
- int AK_index_table_exist (char *indexTblName)

  Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)
- AK_header * AK_get_index_header (char *indexTblName)

  Function that getts index table header.
- void AK_print_index_table (char *indexTblName)

  Function for printing index table.
- void AK_index_test ()

  Test funtion for index structures(list) and printing table.

5.20.1 Detailed Description

Provides functions for indexes

5.20.2 Function Documentation

5.20.2.1 void Ak_Delete_All_elementsAd (list_ad * L)

Function for deleting all nodes in linked list.

Author

Unknown

Parameters

L	list head

Returns

No return value

5.20.2.2 void Ak_Delete_elementAd (element_ad *Deletedelement_op,* list_ad * L)

Function for deleting a node in linked list.

Author

Unknown

Parameters

Deletedelement_op	- address of node to delete
list_ad	*L - list head

Returns

No return value

5.20.2.3 element_ad Ak_Get_First_elementAd (list_ad * L)

Function for finding first node of linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

Address of first node

5.20.2.4 AK_header* AK_get_index_header (char * *indexTblName*)

Function that getts index table header.

Author

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. allocate array
5. copy table header to the array

Parameters

**tblName*	table name

Returns

array of table header

5.20.2.5 int AK_get_index_num_records (char * *indexTblName*)

Determine number of rows in the table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

Parameters

**tableName*	table name

Returns

number of rows in the table

5.20.2.6 struct list_node * AK_get_index_tuple (int *row*, int *column*, char * *indexTblName*)

Function that gets value in some row and column.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

row	zero-based row index
column	zero-based column index
**tblName*	table name

Returns

value in the list

5.20.2.7 element_ad Ak_Get_Last_elementAd (list_ad * *L*)

Function for finding last node of linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

Address of last node or 0 if list is empty

5.20.2.8 element_ad Ak_Get_Next_elementAd (element_ad *Currentelement_op*)

Function for finding the next node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	address of current node

Returns

Address of next node or 0 if current node is last in list

5.20.2.9 int Ak_Get_Position_Of_elementAd (element_ad *Searchedelement_op,* list_ad * *L*)

Function for finding the position of a node in linked list.

Author

Unknown

Parameters

Searchedelement_op	address of current note
**L*	linked list head

Returns

Integer value of current node's order in the list

5.20.2.10 element_ad Ak_Get_Previous_elementAd (element_ad *Currentelement_op,* element_ad *L*)

Function for finding the previous node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	Address of current node
L	previous element

Returns

Address of previous node or 0 if the current node is the head or the list is empty

5.20.2.11 int AK_index_table_exist (char * *indexTblName*)

Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

tblName	table name

Returns

returns 1 if table exist or returns 0 if table does not exist

5.20.2.12 void AK_index_test ( )

Test funtion for index structures(list) and printing table.

Author

Lovro Predovan

Returns

No return value

5.20.2.13 void Ak_InitializelistAd (list_ad * *L*)

Function for initalizing linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

No return value

5.20.2.14 void Ak_Insert_NewelementAd (int *addBlock,* int *indexTd,* char * *attName,* element_ad *elementBefore*)

Function for inserting a new element into linked list.

Author

Unknown

Parameters

addBlock	address block
indexTd	index table destination
∗*attname*	attribute name
elementBefore	address of the node after which the new node will be inserted

Returns

No return value

5.20.2.15 int AK_num_index_attr (char ∗ *indexTblName*)

Function for getting number of elements in index table.

Author

Lovro Predovan

Parameters

index	table name

Returns

No return value

5.20.2.16 void AK_print_index_table (char ∗ *indexTblName*)

Function for printing index table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

∗*tblName*	table name

Returns

No return value

5.21 file/idx/index.h File Reference

```
#include "../../auxi/mempro.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../file/files.h"
```

Include dependency graph for index.h: This graph shows which files directly or indirectly include this file:

Classes

- struct struct_add

  Structure defining node address.
- struct list_structure_ad

Typedefs

- typedef struct list_structure_ad **list_structure_ad**
- typedef list_structure_ad ∗ **element_ad**
- typedef list_structure_ad **list_ad**

Functions

- int AK_index_table_exist (char ∗indexTblName)

  Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)
- void AK_print_index_table (char ∗indexTblName)

  Function for printing index table.
- struct list_node ∗ AK_get_index_tuple (int row, int column, char ∗indexTblName)

  Function that gets value in some row and column.
- int AK_get_index_num_records (char ∗indexTblName)

  Determine number of rows in the table.
- int AK_num_index_attr (char ∗indexTblName)

  Function for getting number of elements in index table.
- void Ak_InitializelistAd (list_ad ∗L)

  Function for initalizing linked list.
- element_ad Ak_Get_First_elementAd (list_ad ∗L)

  Function for finding first node of linked list.
- element_ad Ak_Get_Last_elementAd (list_ad ∗L)

  Function for finding last node of linked list.
- element_ad Ak_Get_Next_elementAd (element_ad Currentelement_op)

  Function for finding the next node of a node in linked list.
- element_ad Ak_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  Function for finding the previous node of a node in linked list.
- int Ak_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad ∗L)

  Function for finding the position of a node in linked list.
- void Ak_Delete_elementAd (element_ad Deletedelement_op, list_ad ∗L)

Function for deleting a node in linked list.

- void Ak_Delete_All_elementsAd (list_ad *L)

  Function for deleting all nodes in linked list.

- void Ak_Insert_NewelementAd (int addBlock, int indexTd, char *attName, element_ad elementBefore)

  Function for inserting a new element into linked list.

- void AK_index_test ()

  Test funtion for index structures(list) and printing table.

5.21.1 Detailed Description

Header file that provides data structures for bitmap index

5.21.2 Function Documentation

5.21.2.1 void Ak_Delete_All_elementsAd (list_ad * *L*)

Function for deleting all nodes in linked list.

Author

Unknown

Parameters

L	list head

Returns

No return value

5.21.2.2 void Ak_Delete_elementAd (element_ad *Deletedelement_op*, list_ad * *L*)

Function for deleting a node in linked list.

Author

Unknown

Parameters

Deletedelement_op	- address of node to delete
list_ad	*L - list head

Returns

No return value

5.21.2.3 element_ad Ak_Get_First_elementAd (list_ad * L)

Function for finding first node of linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

Address of first node

5.21.2.4 int AK_get_index_num_records (char * indexTblName)

Determine number of rows in the table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

Parameters

*tableName	table name

Returns

number of rows in the table

5.21.2.5 struct list_node* AK_get_index_tuple (int row, int column, char * indexTblName)

Function that gets value in some row and column.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

row	zero-based row index
column	zero-based column index
∗*tblName*	table name

Returns

value in the list

5.21.2.6 element_ad Ak_Get_Last_elementAd (list_ad ∗ *L*)

Function for finding last node of linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

Address of last node or 0 if list is empty

5.21.2.7 element_ad Ak_Get_Next_elementAd (element_ad *Currentelement_op*)

Function for finding the next node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	address of current node

Returns

Address of next node or 0 if current node is last in list

5.21.2.8 int Ak_Get_Position_Of_elementAd (element_ad *Searchedelement_op,* list_ad * *L*)

Function for finding the position of a node in linked list.

Author

Unknown

Parameters

Searchedelement_op	address of current note
**L*	linked list head

Returns

Integer value of current node's order in the list

5.21.2.9 element_ad Ak_Get_Previous_elementAd (element_ad *Currentelement_op,* element_ad *L*)

Function for finding the previous node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	Address of current node
L	previous element

Returns

Address of previous node or 0 if the current node is the head or the list is empty

5.21.2.10 int AK_index_table_exist (char * *indexTblName*)

Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

tblName	table name

Returns

returns 1 if table exist or returns 0 if table does not exist

5.21.2.11 void AK_index_test ( )

Test funtion for index structures(list) and printing table.

Author

Lovro Predovan

Returns

No return value

5.21.2.12 void Ak_InitializelistAd (list_ad * *L*)

Function for initalizing linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

No return value

5.21.2.13 void Ak_Insert_NewelementAd (int *addBlock,* int *indexTd,* char * *attName,* element_ad *elementBefore*)

Function for inserting a new element into linked list.

Author

Unknown

Parameters

addBlock	address block
indexTd	index table destination
**attname*	attribute name
elementBefore	address of the node after which the new node will be inserted

Returns

No return value

5.21.2.14 int AK_num_index_attr (char * *indexTblName*)

Function for getting number of elements in index table.

Author

Lovro Predovan

Parameters

index	table name

Returns

No return value

5.21.2.15 void AK_print_index_table (char * *indexTblName*)

Function for printing index table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

**tblName*	table name

Returns

No return value

5.22 file/table.c File Reference

```
#include "../file/table.h"
```

Include dependency graph for table.c:

Functions

- AK_create_table_parameter * **AK_create_create_table_parameter** (int type, char *name)
- void **AK_create_table** (char *tblName, AK_create_table_parameter *parameters, int attribute_count)

- int AK_num_attr (char *tblName)
  Determine the number of attributes in the table.
- int AK_get_num_records (char *tblName)
  Determine number of rows in the table.
- AK_header * AK_get_header (char *tblName)
  Function that getts table header.
- char * AK_get_attr_name (char *tblName, int index)
  Function that gets attribute name for some zero-based index.
- int AK_get_attr_index (char *tblName, char *attrName)
  Function that gets zero-based index for atrribute.
- struct list_node * AK_get_column (int num, char *tblName)
  Function that gets all values in some column and put on the list.
- struct list_node * AK_get_row (int num, char *tblName)
  Function that gets all values in some row and put on the list.
- struct list_node * AK_get_tuple (int row, int column, char *tblName)
  Function that gets value in some row and column.
- char * AK_tuple_to_string (struct list_node *tuple)
  Function that converts tuple value to string.
- void AK_print_row_spacer (int col_len[ ], int length)
  Function that prints row spacer.
- void AK_print_row (int col_len[ ], struct list_node *row)
  Function that prints table row.
- int AK_table_exist (char *tblName)
  Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)
- void AK_print_table (char *tblName)
  Function for printing table.
- void AK_print_row_spacer_to_file (int col_len[ ], int length)
  Function that prints row spacer update by Luka Rajcevic.
- char * get_row_attr_data (int column, struct list_node *node)
  Function that returns value of attribute from row.
- void AK_print_row_to_file (int col_len[ ], struct list_node *row)
  Function that prints table row update by Luka Rajcevic.
- void AK_print_table_to_file (char *tblName)
  Function for printing table.
- int AK_table_empty (char *tblName)
  Function that check whether table is empty.
- int AK_get_table_obj_id (char *table)
  Function that gets obj_id of named table from AK_relation system table.
- int AK_check_tables_scheme (AK_mem_block *tbl1_temp_block, AK_mem_block *tbl2_temp_block, char *operator_name)
  Function to check if tables have the same relation schema.
- int AK_rename (char *old_table_name, char *old_attr, char *new_table_name, char *new_attr)
  Function for renaming table and/or attribute in table (moved from rename.c)
- void AK_table_test ()
  Function for testing table abstraction.
- void AK_op_rename_test ()
  Function for rename operator testing (moved from rename.c)

5.22.1 Detailed Description

Provides functions for table abstraction

5.22.2 Function Documentation

5.22.2.1 int AK_check_tables_scheme (AK_mem_block * *tbl1_temp_block,* AK_mem_block * *tbl2_temp_block,* char * *operator_name*)

Function to check if tables have the same relation schema.

Author

Dino Laktašić, abstracted from difference.c for use in difference.c, intersect.c and union.c by Tomislav Mikulček

Parameters

tbl1_temp_block	first cache block of the first table
tbl2_temp_block	first cache block of the second table
operator_name	the name of operator, used for displaying error message

Returns

if success returns num of attributes in schema, else returns EXIT_ERROR

5.22.2.2 int AK_get_attr_index (char * *tblName,* char * *attrName*)

Function that gets zero-based index for atrribute.

Author

Matija Šestak.

Parameters

**tblName*	table name
**attrName*	attribute name

Returns

zero-based index

5.22.2.3 char* AK_get_attr_name (char * *tblName,* int *index*)

Function that gets attribute name for some zero-based index.

Author

Matija Šestak.

Parameters

∗*tblName*	table name
index	zero-based index

Returns

attribute name

5.22.2.4 struct list_node∗ AK_get_column (int *num,* char ∗ *tblName*)

Function that gets all values in some column and put on the list.

Author

Matija Šestak.

Parameters

num	zero-based column index
∗*tblName*	table name

Returns

column values list

5.22.2.5 AK_header∗ AK_get_header (char ∗ *tblName*)

Function that getts table header.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. allocate array
5. copy table header to the array

Parameters

∗*tblName*	table name

Returns

array of table header

5.22.2.6 int AK_get_num_records (char * *tblName*)

Determine number of rows in the table.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

Parameters

**tableName*	table name

Returns

number of rows in the table

5.22.2.7 struct list_node* AK_get_row (int *num,* char * *tblName*)

Function that gets all values in some row and put on the list.

Author

Markus Schatten, Matija Šestak.

Parameters

num	zero-based row index
*	tblName table name

Returns

row values list

5.22.2.8 int AK_get_table_obj_id (char * *table*)

Function that gets obj_id of named table from AK_relation system table.

Author

Dejan Frankovic

Parameters

**table*	table name

Returns

obj_id of the table or EXIT_ERROR if there is no table with that name

5.22.2.9 struct list_node* AK_get_tuple (int *row,* int *column,* char * *tblName*)

Function that gets value in some row and column.

Author

Matija Šestak.

Parameters

row	zero-based row index
column	zero-based column index
**tblName*	table name

Returns

value in the list

5.22.2.10 int AK_num_attr (char * *tblName*)

Determine the number of attributes in the table.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. while header tuple exists in the block, increment num_attr

Parameters

*	tblName table name

Returns

number of attributes in the table

5.22.2.11 void AK_op_rename_test ( )

Function for rename operator testing (moved from rename.c)

Author

Mislav Čakarić, edited by Ljubo Barać

Returns

No return value

5.22.2.12 void AK_print_row (int *col_len[ ],* struct list_node * *row*)

Function that prints table row.

Author

Dino Laktašić

Parameters

col_len[ ]	array of max lengths for each attribute
**row*	list with row elements

Returns

No return value

5.22.2.13 void AK_print_row_spacer (int *col_len[ ],* int *length*)

Function that prints row spacer.

Author

Dino Laktašić.

Parameters

col_len[ ]	max lengths for each attribute cell
length	total table width

Returns

printed row spacer

5.22.2.14 void AK_print_row_spacer_to_file (int *col_len[ ],* int *length*)

Function that prints row spacer update by Luka Rajcevic.

Author

Dino Laktašić.

Parameters

col_len[ ]	max lengths for each attribute cell
length	total table width

Returns

printed row spacer

5.22.2.15 void AK_print_row_to_file (int *col_len[ ],* struct list_node * *row*)

Function that prints table row update by Luka Rajcevic.

Author

Dino Laktašić

Parameters

col_len[ ]	array of max lengths for each attribute
**row*	list with row elements

Returns

No return value

5.22.2.16 void AK_print_table (char * *tblName*)

Function for printing table.

Author

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one)

Parameters

∗*tblName*	table name

Returns

No return value

5.22.2.17 void AK_print_table_to_file (char ∗ *tblName*)

Function for printing table.

Author

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

Parameters

∗*tblName*	table name

Returns

No return value update by Anto Tomaš (corrected the Ak_DeleteAll_L3 function)

5.22.2.18 int AK_rename (char ∗ *old_table_name,* char ∗ *old_attr,* char ∗ *new_table_name,* char ∗ *new_attr*)

Function for renaming table and/or attribute in table (moved from rename.c)

Author

Mislav Čakarić edited by Ljubo Barać

Parameters

old_table_name	old name of the table
new_table_name	new name of the table
old_attr	name of the attribute to rename
new_attr	new name for the attribute to rename

Returns

EXIT_ERROR or EXIT_SUCCESS

5.22.2.19 int AK_table_empty (char * tblName)

Function that check whether table is empty.

Author

Matija Šestak.

Parameters

*tblName	table name

Returns

true/false

5.22.2.20 int AK_table_exist (char * tblName)

Function examines whether there is a table with the name "tblName" in the system catalog (AK_relation)

Author

Jurica Hlevnjak

Parameters

tblName	table name

Returns

returns 1 if table exist or returns 0 if table does not exist

5.22.2.21 void AK_table_test ( )

Function for testing table abstraction.

Author

Unknown

Returns

No return value

by Ana-Marija Balen - added getRow function to the test

5.22.2.22 char* AK_tuple_to_string (struct list_node * *tuple*)

Function that converts tuple value to string.

Author

Matija Šestak.

Parameters

**tuple*	tuple in the list

Returns

tuple value as a string

5.22.2.23 char* get_row_attr_data (int *column*, struct list_node * *node*)

Function that returns value of attribute from row.

Author

Leon Palaić

Parameters

column	index of column atribute
**row*	list with row elements

Returns

atribute data

5.23 file/table.h File Reference

```
#include "../mm/memoman.h"
#include "../auxi/mempro.h"
#include <time.h>
```

Include dependency graph for table.h: This graph shows which files directly or indirectly include this file:

Classes

- struct AK_create_table_struct

Typedefs

- typedef struct AK_create_table_struct **AK_create_table_parameter**

Functions

- AK_create_table_parameter * **AK_create_create_table_parameter** (int type, char *name)
- void **AK_create_table** (char *tblName, AK_create_table_parameter *parameters, int attribute_count)
- int AK_num_attr (char *tblName)

  Determine the number of attributes in the table.
- int AK_get_num_records (char *tblName)

  Determine number of rows in the table.
- AK_header * AK_get_header (char *tblName)

  Function that getts table header.
- char * AK_get_attr_name (char *tblName, int index)

  Function that gets attribute name for some zero-based index.
- int AK_get_attr_index (char *tblName, char *attrName)

  Function that gets zero-based index for atrribute.
- struct list_node * AK_get_column (int num, char *tblName)

  Function that gets all values in some column and put on the list.
- struct list_node * AK_get_row (int num, char *tblName)

  Function that gets all values in some row and put on the list.
- struct list_node * AK_get_tuple (int row, int column, char *tblName)

  Function that gets value in some row and column.
- char * AK_tuple_to_string (struct list_node *tuple)

  Function that converts tuple value to string.
- void AK_print_row_spacer (int col_len[ ], int length)

  Function that prints row spacer.
- void AK_print_row (int col_len[ ], struct list_node *row)

  Function that prints table row.
- void AK_print_table (char *tblName)

  Function for printing table.
- void AK_print_row_spacer_to_file (int col_len[ ], int length)

  Function that prints row spacer update by Luka Rajcevic.
- void AK_print_row_to_file (int col_len[ ], struct list_node *row)

  Function that prints table row update by Luka Rajcevic.
- void AK_print_table_to_file (char *tblName)

  Function for printing table.
- int AK_table_empty (char *tblName)

  Function that check whether table is empty.
- int AK_get_table_obj_id (char *table)

  Function that gets obj_id of named table from AK_relation system table.
- int AK_check_tables_scheme (AK_mem_block *tbl1_temp_block, AK_mem_block *tbl2_temp_block, char *operator_name)

  Function to check if tables have the same relation schema.
- char * **get_row_attr_data** ()
- struct list_node * **AK_get_table_row** (int num, char *tblName)
- void AK_table_test ()

  Function for testing table abstraction.
- int AK_rename (char *old_table_name, char *old_attr, char *new_table_name, char *new_attr)

  Function for renaming table and/or attribute in table (moved from rename.c)
- void AK_op_rename_test ()

  Function for rename operator testing (moved from rename.c)

5.23.1 Detailed Description

Header file that provides data structures for table abstraction

5.23.2 Function Documentation

5.23.2.1 int AK_check_tables_scheme (AK_mem_block * *tbl1_temp_block,* AK_mem_block * *tbl2_temp_block,* char * *operator_name*)

Function to check if tables have the same relation schema.

Author

Dino Laktašić, abstracted from difference.c for use in difference.c, intersect.c and union.c by Tomislav Mikulček

Parameters

tbl1_temp_block	first cache block of the first table
tbl2_temp_block	first cache block of the second table
operator_name	the name of operator, used for displaying error message

Returns

if success returns num of attributes in schema, else returns EXIT_ERROR

5.23.2.2 int AK_get_attr_index (char * *tblName,* char * *attrName*)

Function that gets zero-based index for atrribute.

Author

Matija Šestak.

Parameters

**tblName*	table name
**attrName*	attribute name

Returns

zero-based index

5.23.2.3 char* AK_get_attr_name (char * *tblName,* int *index*)

Function that gets attribute name for some zero-based index.

Author

Matija Šestak.

Parameters

**tblName*	table name
index	zero-based index

Returns

attribute name

5.23.2.4 struct list_node* AK_get_column (int *num,* char * *tblName*)

Function that gets all values in some column and put on the list.

Author

Matija Šestak.

Parameters

num	zero-based column index
**tblName*	table name

Returns

column values list

5.23.2.5 AK_header* AK_get_header (char * *tblName*)

Function that getts table header.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. allocate array
5. copy table header to the array

Parameters

∗*tblName*	table name

Returns

array of table header

5.23.2.6 int AK_get_num_records (char ∗ *tblName*)

Determine number of rows in the table.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

Parameters

∗*tableName*	table name

Returns

number of rows in the table

5.23.2.7 struct list_node∗ AK_get_row (int *num,* char ∗ *tblName*)

Function that gets all values in some row and put on the list.

Author

Markus Schatten, Matija Šestak.

Parameters

num	zero-based row index
*	tblName table name

Returns

row values list

5.23.2.8 int AK_get_table_obj_id (char * *table*)

Function that gets obj_id of named table from AK_relation system table.

Author

Dejan Frankovic

Parameters

**table*	table name

Returns

obj_id of the table or EXIT_ERROR if there is no table with that name

5.23.2.9 struct list_node* AK_get_tuple (int *row,* int *column,* char * *tblName*)

Function that gets value in some row and column.

Author

Matija Šestak.

Parameters

row	zero-based row index
column	zero-based column index
**tblName*	table name

Returns

value in the list

5.23.2.10 int AK_num_attr (char * *tblName*)

Determine the number of attributes in the table.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. while header tuple exists in the block, increment num_attr

Parameters

*	tblName table name

Returns

number of attributes in the table

5.23.2.11 void AK_op_rename_test ( )

Function for rename operator testing (moved from rename.c)

Author

Mislav Čakarić, edited by Ljubo Barać

Returns

No return value

5.23.2.12 void AK_print_row (int *col_len[ ],* struct list_node * *row*)

Function that prints table row.

Author

Dino Laktašić

Parameters

col_len[ ]	array of max lengths for each attribute
**row*	list with row elements

Returns

No return value

5.23.2.13 void AK_print_row_spacer (int *col_len[ ],* int *length*)

Function that prints row spacer.

Author

Dino Laktašić.

Parameters

col_len[ ]	max lengths for each attribute cell
length	total table width

Returns

printed row spacer

5.23.2.14 void AK_print_row_spacer_to_file (int *col_len[ ],* int *length*)

Function that prints row spacer update by Luka Rajcevic.

Author

Dino Laktašić.

Parameters

col_len[ ]	max lengths for each attribute cell
length	total table width

Returns

printed row spacer

5.23.2.15 void AK_print_row_to_file (int *col_len[ ],* struct list_node * *row*)

Function that prints table row update by Luka Rajcevic.

Author

Dino Laktašić

Parameters

col_len[ ]	array of max lengths for each attribute
**row*	list with row elements

Returns

No return value

5.23.2.16 void AK_print_table (char * *tblName*)

Function for printing table.

Author

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one)

Parameters

**tblName*	table name

Returns

No return value

5.23.2.17 void AK_print_table_to_file (char * *tblName*)

Function for printing table.

Author

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

Parameters

**tblName*	table name

Returns

No return value update by Anto Tomaš (corrected the Ak_DeleteAll_L3 function)

5.23.2.18 int AK_rename (char * *old_table_name,* char * *old_attr,* char * *new_table_name,* char * *new_attr*)

Function for renaming table and/or attribute in table (moved from rename.c)

Author

Mislav Čakarić edited by Ljubo Barać

Parameters

old_table_name	old name of the table
new_table_name	new name of the table
old_attr	name of the attribute to rename
new_attr	new name for the attribute to rename

Returns

EXIT_ERROR or EXIT_SUCCESS

5.23.2.19 int AK_table_empty (char * *tblName*)

Function that check whether table is empty.

Author

Matija Šestak.

Parameters

*tblName	table name

Returns

true/false

5.23.2.20 void AK_table_test ( )

Function for testing table abstraction.

Author

Unknown

Returns

No return value

by Ana-Marija Balen - added getRow function to the test

5.23.2.21 char* AK_tuple_to_string (struct list_node * *tuple*)

Function that converts tuple value to string.

Author

Matija Šestak.

Parameters

∗tuple	tuple in the list

Returns

tuple value as a string

5.24 file/test.c File Reference

```
#include <pthread.h>
#include <stdio.h>
#include "test.h"
#include "../trans/transaction.h"
#include "../file/table.h"
#include "../auxi/auxiliary.h"
#include "../opti/rel_eq_comut.h"
```

Include dependency graph for test.c:

Functions

- char ∗ AK_get_table_atribute_types (char ∗tblName)
  returns a string containing attribute types for supplied table name, seperated by ATTR_DELIMITER
- int create_header_test (char ∗tbl_name, char ∗∗attr_name, int _num, int ∗_type)
  Function for creating test table header.
- int insert_data_test (char ∗tbl_name, char ∗∗attr_name, char ∗∗attr_value, int _num, int ∗_type)
  Function for inserting test data into table (needed for python testing)
- int selection_test (char ∗src_table, char ∗dest_table, char ∗∗sel_query, int _num, int ∗_type)
  Function for selection operator on one table.
- int get_column_test (int num, char ∗tbl)
  prints requested column
- int get_row_test (int num, char ∗tbl)
  prints requested row
- void AK_create_test_tables ()
  Function for creating test tables.

5.24.1 Detailed Description

Provides functions for testing purposes

5.24.2 Function Documentation

5.24.2.1 void AK_create_test_tables ( )

Function for creating test tables.

Author

Dino Laktašić

Returns

No return value

5.24.2.2 char* AK_get_table_atribute_types (char * *tblName*)

returns a string containing attribute types for supplied table name, seperated by ATTR_DELIMITER

Author

Goran Štrok

Parameters

tblName	name of the table for which the attribute types will be returned

5.24.2.3 int create_header_test (char * *tbl_name,* char ** *attr_name,* int *_num,* int * *_type*)

Function for creating test table header.

Author

Luka Rajcevic

Parameters

tbl_name	- name of the table for which the header will be created
attr_name	- array of attribute names
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

1 if ok, 0 otherwise

5.24.2.4 int get_column_test (int *num,* char * *tbl*)

prints requested column

Author

Luka Rajcevic

Returns

1 if column is found, 0 otherwise

Parameters

num	- 0 based index of column
tbl	- name of the table

5.24.2.5 int get_row_test (int *num,* char * *tbl*)

prints requested row

Author

Luka Rajcevic

Returns

1 if row is found, 0 otherwise

Parameters

num	- 0 based index of row
tbl	- name of the table

5.24.2.6 int insert_data_test (char * *tbl_name,* char ** *attr_name,* char ** *attr_value,* int *_num,* int * *_type*)

Function for inserting test data into table (needed for python testing)

Author

Luka Rajcevic

Parameters

tbl_name	- name of the table for which the header will be created
attr_name	- array of attribute names
attr_value	- values of attributes to be inserted
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.24.2.7 int selection_test (char * *src_table,* char * *dest_table,* char ** *sel_query,* int *_num,* int * *_type*)

Function for selection operator on one table.

Author

Luka Rajcevic

-

Parameters

src_table	- name of the source table •
dest_table	- table in which selection will be stored
sel_query	- array of operators, operands and attributes (postfix query)
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.25 file/test.h File Reference

```
#include "files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for test.h: This graph shows which files directly or indirectly include this file:

Functions

- char * AK_get_table_atribute_types (char *tblName)

  returns a string containing attribute types for supplied table name, seperated by ATTR_DELIMITER
- int create_header_test (char *tbl_name, char **attr_name, int _num, int *_type)

  Function for creating test table header.
- int insert_data_test (char *tbl_name, char **attr_name, char **attr_value, int _num, int *_type)

  Function for inserting test data into table (needed for python testing)
- int selection_test (char *src_table, char *dest_table, char **sel_query, int _num, int *_type)

  Function for selection operator on one table.
- int get_column_test (int num, char *tbl)

  prints requested column
- int get_row_test (int num, char *tbl)

  prints requested row
- void AK_create_test_tables ()

  Function for creating test tables.

5.25.1 Detailed Description

Header file that provides functions for testing purposes

5.25.2 Function Documentation

5.25.2.1 void AK_create_test_tables ( )

Function for creating test tables.

Author

Dino Laktašić

Returns

No return value

5.25.2.2 char* AK_get_table_atribute_types (char * *tblName*)

returns a string containing attribute types for supplied table name, seperated by ATTR_DELIMITER

Author

Goran Štrok

Parameters

tblName	name of the table for which the attribute types will be returned

5.25.2.3 int create_header_test (char * *tbl_name,* char ** *attr_name,* int *_num,* int * *_type*)

Function for creating test table header.

Author

Luka Rajcevic

Parameters

tbl_name	- name of the table for which the header will be created
attr_name	- array of attribute names
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

1 if ok, 0 otherwise

5.25.2.4 int get_column_test (int *num,* char * *tbl*)

prints requested column

Author

Luka Rajcevic

Returns

1 if column is found, 0 otherwise

Parameters

num	- 0 based index of column
tbl	- name of the table

5.25.2.5 int get_row_test (int *num,* char * *tbl*)

prints requested row

Author

Luka Rajcevic

Returns

1 if row is found, 0 otherwise

Parameters

num	- 0 based index of row
tbl	- name of the table

5.25.2.6 int insert_data_test (char * *tbl_name,* char ** *attr_name,* char ** *attr_value,* int *_num,* int * *_type*)

Function for inserting test data into table (needed for python testing)

Author

Luka Rajcevic

Parameters

tbl_name	- name of the table for which the header will be created
attr_name	- array of attribute names
attr_value	- values of attributes to be inserted
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.25.2.7 int selection_test (char * *src_table*, char * *dest_table*, char ** *sel_query*, int *_num*, int * *_type*)

Function for selection operator on one table.

Author

Luka Rajcevic

-

Parameters

src_table	- name of the source table •
dest_table	- table in which selection will be stored
sel_query	- array of operators, operands and attributes (postfix query)
_num	- number of attributes
_type	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.26 mm/memoman.c File Reference

```
#include "memoman.h"
```

Include dependency graph for memoman.c:

Functions

- int AK_cache_block (int num, AK_mem_block *mem_block)

  Function caches block into memory.
- int AK_cache_AK_malloc ()

  Function initializes the global cache memory (variable db_cache)
- int AK_redo_log_AK_malloc ()

  Function initializes the global redo log memory (variable redo_log)
- int AK_find_available_result_block ()

  Function find available block for result caching in circular array.
- unsigned long AK_generate_result_id (unsigned char *str)

  Generate unique hash identifier for each cached result by using djb2 algorithm.
- void AK_cache_result (char *srcTable, AK_block *temp_block, AK_header header[ ])

  Cache fetched result block in memory.
- int AK_query_mem_AK_malloc ()

Function initializes the global query memory (variable query_mem)

- int AK_memoman_init ()

  Function initializes memory manager (cache, redo log and query memory)

- AK_mem_block * AK_get_block (int num)

  Function reads a block from memory. If the block is cached returns the cached block. Else uses AK_cache_block to read the block to cache and then returns it.

- void AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  Modify the "dirty" bit of a block, and update timestamps accordingly.

- int AK_refresh_cache ()

  Function re-read all the blocks from disk.

- table_addresses * AK_get_index_segment_addresses (char *segmentName)

  Function for geting addresses of some table.

- table_addresses * AK_get_segment_addresses (char *segmentName)

  Function for geting addresses of some table.

- table_addresses * AK_get_table_addresses (char *table)

  function for geting addresses of some table

- table_addresses * AK_get_index_addresses (char *index)

  Function for geting addresses of some index.

- int AK_find_AK_free_space (table_addresses *addresses)

  Function to find AK_free space in some block betwen block addresses. It's made for insert_row()

- int AK_init_new_extent (char *table_name, int extent_type)

  Function that extends the segment.

- int AK_flush_cache ()

  Function that flushes memory blocks to disk file.

- void **AK_memoman_test** ()
- void **AK_memoman_test2** ()

5.26.1 Detailed Description

Defines functions for the memory manager of Kalashnikov DB

5.26.2 Function Documentation

5.26.2.1 int AK_cache_AK_malloc ( )

Function initializes the global cache memory (variable db_cache)

Author

Markus Schatten, Matija Šestak(revised)

Returns

EXIT_SUCCESS if the cache memory has been initialized, EXIT_ERROR otherwise

5.26.2.2 int AK_cache_block (int *num*, AK_mem_block * *mem_block*)

Function caches block into memory.

Author

Nikola Bakoš, Matija Šestak(revised)

Parameters

num	block number (address)
mem_block	address of memmory block

Returns

EXIT_SUCCESS if the block has been successfully read into memory, EXIT_ERROR otherwise

read the block from the given address

set dirty bit in mem_block struct

get the timestamp

set timestamp_read

set timestamp_last_change

5.26.2.3 void AK_cache_result (char * *srcTable,* AK_block * *temp_block,* AK_header *header[ ]*)

Cache fetched result block in memory.

Author

Mario Novoselec

5.26.2.4 int AK_find_AK_free_space (table_addresses * *addresses*)

Function to find AK_free space in some block betwen block addresses. It's made for insert_row()

Author

Matija Novak, updated by Matija Šestak(function now uses caching)

Parameters

address	addresses of extents

Returns

address of the block to write in

5.26.2.5 int AK_find_available_result_block ( )

Function find available block for result caching in circular array.

Author

Mario Novoselec

Returns

available_index

5.26.2.6 int AK_flush_cache ( )

Function that flushes memory blocks to disk file.

Author

Matija Šestak

Returns

EXIT_SUCCESS

if block form cache can not be writed to DB file -> EXIT_ERROR

5.26.2.7 unsigned long AK_generate_result_id (unsigned char * *str*)

Generate unique hash identifier for each cached result by using djb2 algorithm.

Author

Mario Novoselec

Returns

hash

5.26.2.8 AK_mem_block* AK_get_block (int *num*)

Function reads a block from memory. If the block is cached returns the cached block. Else uses AK_cache_block to read the block to cache and then returns it.

Author

Tomislav Fotak, updated by Matija Šestak

Parameters

num	block number (address)

Returns

segment start address

if block form cache can not be writed to DB file -> EXIT_ERROR

if block form cache can not be writed to DB file -> EXIT_ERROR

5.26.2.9 table_addresses* AK_get_index_addresses (char * *index*)

Function for geting addresses of some index.

Author

Mislav Čakarić

Parameters

index	index name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.26.2.10 table_addresses* AK_get_index_segment_addresses (char * *segmentName*)

Function for geting addresses of some table.

Author

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić,Lovro Predovan

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.26.2.11 table_addresses* AK_get_segment_addresses (char * *segmentName*)

Function for geting addresses of some table.

Author

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.26.2.12 table_addresses* AK_get_table_addresses (char * *table*)

function for geting addresses of some table

Author

Mislav Čakarić

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.26.2.13 int AK_init_new_extent (char * *table_name,* int *extent_type*)

Function that extends the segment.

Author

Nikola Bakoš, updated by Matija Šestak (function now uses caching), updated by Mislav Čakarić, updated by Dino Laktašić

Parameters

table_name	name of segment to extent
extent_type	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP

Returns

address of new extent, otherwise EXIT_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

5.26.2.14 void AK_mem_block_modify (AK_mem_block * *mem_block,* int *dirty*)

Modify the "dirty" bit of a block, and update timestamps accordingly.

Author

Alen Novosel.

5.26.2.15 int AK_memoman_init ( )

Function initializes memory manager (cache, redo log and query memory)

Author

Miroslav Policki

Returns

EXIT_SUCCESS if the query memory manager has been initialized, EXIT_ERROR otherwise

5.26.2.16 int AK_query_mem_AK_malloc ( )

Function initializes the global query memory (variable query_mem)

Author

Matija Novak

Returns

EXIT_SUCCESS if the query memory has been initialized, EXIT_ERROR otherwise

allocate memory for global variable query_mem

allocate memory for variable query_mem_lib which is used in query_mem->parsed

allocate memory for variable query_mem_dict which is used in query_mem->dictionary

allocate memory for variable query_mem_result which is used in query_mem->result

allocate memory for variable tuple_dict which is used in query_mem->dictionary->dictionary[]

5.26.2.17 int AK_redo_log_AK_malloc ( )

Function initializes the global redo log memory (variable redo_log)

Author

Dejan Sambolić updated by Dražen Bandić, updated by Tomislav Turek

Returns

EXIT_SUCCESS if the redo log memory has been initialized, EXIT_ERROR otherwise

5.26.2.18 int AK_refresh_cache ( )

Function re-read all the blocks from disk.

Author

Matija Šestak.

Returns

EXIT_SUCCESS

5.27 mm/memoman.h File Reference

```
#include "../dm/dbman.h"
#include "../auxi/mempro.h"
```

Include dependency graph for memoman.h: This graph shows which files directly or indirectly include this file:

Classes

- struct AK_mem_block
  Structure that defines a block of data in memory.
- struct AK_db_cache
  Structure that defines global cache memory.
- struct AK_command_recovery_struct
  recovery structure used to recover commands from binary file
- struct AK_redo_log
  Structure that defines global redo log.
- struct AK_query_mem_lib
  Structure that defines global query memory for libraries.
- struct AK_query_mem_dict
  Structure that defines global query memory for data dictionaries.
- struct AK_results
  Structure used for in-memory result caching.
- struct AK_query_mem_result
  Structure that defines global query memory for results.
- struct AK_query_mem
  Structure that defines global query memory.

Functions

- void AK_cache_result (char *srcTable, AK_block *temp_block, AK_header header[ ])

  Cache fetched result block in memory.
- int AK_find_available_result_block ()

  Function find available block for result caching in circular array.
- unsigned long AK_generate_result_id (unsigned char *str)

  Generate unique hash identifier for each cached result by using djb2 algorithm.
- int AK_cache_block (int num, AK_mem_block *mem_block)

  Function caches block into memory.
- int AK_cache_AK_malloc ()

  Function initializes the global cache memory (variable db_cache)
- int AK_redo_log_AK_malloc ()

  Function initializes the global redo log memory (variable redo_log)
- int AK_query_mem_AK_malloc ()

  Function initializes the global query memory (variable query_mem)
- int AK_memoman_init ()

  Function initializes memory manager (cache, redo log and query memory)
- AK_mem_block * AK_get_block (int num)

  Function reads a block from memory. If the block is cached returns the cached block. Else uses AK_cache_block to read the block to cache and then returns it.
- void AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  Modify the "dirty" bit of a block, and update timestamps accordingly.
- int AK_refresh_cache ()

  Function re-read all the blocks from disk.
- table_addresses * AK_get_segment_addresses (char *segmentName)

  Function for geting addresses of some table.
- table_addresses * AK_get_index_segment_addresses (char *segmentName)

  Function for geting addresses of some table.
- table_addresses * AK_get_table_addresses (char *table)

  function for geting addresses of some table
- table_addresses * AK_get_index_addresses (char *index)

  Function for geting addresses of some index.
- int AK_find_AK_free_space (table_addresses *addresses)

  Function to find AK_free space in some block betwen block addresses. It's made for insert_row()
- int AK_init_new_extent (char *table_name, int extent_type)

  Function that extends the segment.
- int AK_flush_cache ()

  Function that flushes memory blocks to disk file.
- void **AK_memoman_test** ()
- void **AK_memoman_test2** ()

Variables

- AK_db_cache * db_cache

  Variable that defines the db cache.
- AK_redo_log * redo_log

  Variable that defines the global redo log.
- AK_query_mem * query_mem

  Variable that defines the global query memory.

5.27.1 Detailed Description

Header file that defines includes and datastructures for the memory manager of Kalashnikov DB

5.27.2 Function Documentation

5.27.2.1 int AK_cache_AK_malloc ( )

Function initializes the global cache memory (variable db_cache)

Author

Markus Schatten, Matija Šestak(revised)

Returns

EXIT_SUCCESS if the cache memory has been initialized, EXIT_ERROR otherwise

5.27.2.2 int AK_cache_block (int *num,* AK_mem_block * *mem_block*)

Function caches block into memory.

Author

Nikola Bakoš, Matija Šestak(revised)

Parameters

num	block number (address)
mem_block	address of memmory block

Returns

EXIT_SUCCESS if the block has been successfully read into memory, EXIT_ERROR otherwise

read the block from the given address

set dirty bit in mem_block struct

get the timestamp

set timestamp_read

set timestamp_last_change

5.27.2.3 void AK_cache_result (char * *srcTable,* AK_block * *temp_block,* AK_header *header[ ]*)

Cache fetched result block in memory.

Author

Mario Novoselec

5.27.2.4 int AK_find_AK_free_space (table_addresses * *addresses*)

Function to find AK_free space in some block betwen block addresses. It's made for insert_row()

Author

Matija Novak, updated by Matija Šestak(function now uses caching)

Parameters

address	addresses of extents

Returns

address of the block to write in

5.27.2.5 int AK_find_available_result_block ( )

Function find available block for result caching in circular array.

Author

Mario Novoselec

Returns

available_index

5.27.2.6 int AK_flush_cache ( )

Function that flushes memory blocks to disk file.

Author

Matija Šestak

Returns

EXIT_SUCCESS

if block form cache can not be writed to DB file -> EXIT_ERROR

5.27.2.7 unsigned long AK_generate_result_id (unsigned char * *str*)

Generate unique hash identifier for each cached result by using djb2 algorithm.

Author

Mario Novoselec

Returns

hash

5.27.2.8 AK_mem_block* AK_get_block (int *num*)

Function reads a block from memory. If the block is cached returns the cached block. Else uses AK_cache_block to read the block to cache and then returns it.

Author

Tomislav Fotak, updated by Matija Šestak

Parameters

num	block number (address)

Returns

segment start address

if block form cache can not be writed to DB file -> EXIT_ERROR

if block form cache can not be writed to DB file -> EXIT_ERROR

5.27.2.9 table_addresses* AK_get_index_addresses (char * *index*)

Function for geting addresses of some index.

Author

Mislav Čakarić

Parameters

index	index name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.27.2.10 table_addresses* AK_get_index_segment_addresses (char * *segmentName*)

Function for geting addresses of some table.

Author

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić,Lovro Predovan

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.27.2.11 table_addresses* AK_get_segment_addresses (char * *segmentName*)

Function for geting addresses of some table.

Author

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.27.2.12 table_addresses* AK_get_table_addresses (char * *table*)

function for geting addresses of some table

Author

Mislav Čakarić

Parameters

table	table name that you search for

Returns

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

5.27.2.13 int AK_init_new_extent (char * *table_name,* int *extent_type*)

Function that extends the segment.

Author

Nikola Bakoš, updated by Matija Šestak (function now uses caching), updated by Mislav Čakarić, updated by Dino Laktašić

Parameters

table_name	name of segment to extent
extent_type	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP

Returns

address of new extent, otherwise EXIT_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

5.27.2.14 void AK_mem_block_modify (AK_mem_block * *mem_block,* int *dirty*)

Modify the "dirty" bit of a block, and update timestamps accordingly.

Author

Alen Novosel.

5.27.2.15 int AK_memoman_init ( )

Function initializes memory manager (cache, redo log and query memory)

Author

Miroslav Policki

Returns

EXIT_SUCCESS if the query memory manager has been initialized, EXIT_ERROR otherwise

5.27.2.16 int AK_query_mem_AK_malloc ( )

Function initializes the global query memory (variable query_mem)

Author

Matija Novak

Returns

EXIT_SUCCESS if the query memory has been initialized, EXIT_ERROR otherwise

allocate memory for global variable query_mem

allocate memory for variable query_mem_lib which is used in query_mem->parsed

allocate memory for variable query_mem_dict which is used in query_mem->dictionary

allocate memory for variable query_mem_result which is used in query_mem->result

allocate memory for variable tuple_dict which is used in query_mem->dictionary->dictionary[]

5.27.2.17 int AK_redo_log_AK_malloc ( )

Function initializes the global redo log memory (variable redo_log)

Author

Dejan Sambolić updated by Dražen Bandić, updated by Tomislav Turek

Returns

EXIT_SUCCESS if the redo log memory has been initialized, EXIT_ERROR otherwise

5.27.2.18 int AK_refresh_cache ( )

Function re-read all the blocks from disk.

Author

Matija Šestak.

Returns

EXIT_SUCCESS

5.28 opti/query_optimization.c File Reference

```
#include "query_optimization.h"
```
Include dependency graph for query_optimization.c:

Functions

- void AK_print_optimized_query (struct list_node *list_query)

  Print optimization table for testing purposes.
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.
- struct list_node * AK_query_optimization (struct list_node *list_query, const char *FLAGS, const int DIFF↩ _PLANS)

  Execute all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.
- void AK_query_optimization_test ()

5.28.1 Detailed Description

Provides functions for general query optimization

5.28.2 Function Documentation

5.28.2.1 struct list_node* AK_execute_rel_eq (struct list_node * *list_query*, const char *rel_eq*, const char * *FLAGS*)

Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.

Author

Dino Laktašić.

Parameters

**list_query*	RA expresion list where we need to apply relational equivalences rules
rel_eq	rel_eq to execute
**FLAGS*	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns struct list_node (RA expresion list) optimized by given relational equivalence rule

5.28.2.2 void AK_print_optimized_query (struct list_node * *list_query*)

Print optimization table for testing purposes.

Author

Dino Laktašić.

Parameters

∗*list_query*	optimized RA expresion list

Returns

list output

5.28.2.3 struct list_node∗ AK_query_optimization (struct list_node ∗ *list_query,* const char ∗ *FLAGS,* const int *DIFF_PLANS*)

Execute all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

Author

Dino Laktašić.

Parameters

∗*list_query*	RA expresion list where we need to apply relational equivalences rules
∗*FLAGS*	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns AK_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

5.28.2.4 void AK_query_optimization_test ( )

Author

Dino Laktašić

Parameters

∗*list_query*	query to be optimized

Returns

No return value

5.29 opti/query_optimization.h File Reference

```
#include "rel_eq_comut.h"
#include "rel_eq_assoc.h"
#include "rel_eq_projection.h"
#include "rel_eq_selection.h"
#include "../auxi/mempro.h"
#include "../sql/view.h"
```

Include dependency graph for query_optimization.h: This graph shows which files directly or indirectly include this file:

Macros

- #define MAX_PERMUTATION 24

  Constant declaring maximum number of permutations.

Functions

- void AK_print_optimized_query (struct list_node *list_query)

  Print optimization table for testing purposes.
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.
- struct list_node * AK_query_optimization (struct list_node *list_query, const char *FLAGS, const int DIFF↩ _PLANS)

  Execute all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.
- void AK_query_optimization_test ()

5.29.1 Detailed Description

Header file that provides functions for general query optimization

5.29.2 Function Documentation

5.29.2.1 struct list_node* AK_execute_rel_eq (struct list_node * *list_query*, const char *rel_eq*, const char * *FLAGS*)

Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.

Author

Dino Laktašić.

Parameters

∗*list_query*	RA expresion list where we need to apply relational equivalences rules
rel_eq	rel_eq to execute
∗*FLAGS*	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns struct list_node (RA expresion list) optimized by given relational equivalence rule

5.29.2.2 void AK_print_optimized_query (struct list_node ∗ *list_query*)

Print optimization table for testing purposes.

Author

Dino Laktašić.

Parameters

∗*list_query*	optimized RA expresion list

Returns

list output

5.29.2.3 struct list_node∗ AK_query_optimization (struct list_node ∗ *list_query,* const char ∗ *FLAGS,* const int *DIFF_PLANS*)

Execute all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

Author

Dino Laktašić.

Parameters

∗*list_query*	RA expresion list where we need to apply relational equivalences rules
∗*FLAGS*	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns AK_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

5.29.2.4 void AK_query_optimization_test ( )

Author

Dino Laktašić

Parameters

∗list_query	query to be optimized

Returns

No return value

5.30 opti/rel_eq_assoc.c File Reference

```
#include "rel_eq_assoc.h"
#include "rel_eq_projection.h"
```

Include dependency graph for rel_eq_assoc.c:

Functions

- int AK_compare (const void ∗a, const void ∗b)
  Function for Struct cost_eval comparison.
- struct list_node ∗ AK_rel_eq_assoc (struct list_node ∗list_rel_eq)
  Main function for generating RA expresion according to associativity equivalence rules.
- void AK_print_rel_eq_assoc (struct list_node ∗list_rel_eq)
  Function for printing RA expresion struct list_node.
- void AK_rel_eq_assoc_test ()
  Function for testing relational equivalences regarding associativity.

5.30.1 Detailed Description

Provides functions for for relational equivalences regarding associativity

5.30.2 Function Documentation

5.30.2.1 int AK_compare (const void ∗ *a*, const void ∗ *b*)

Function for Struct cost_eval comparison.

Author

Dino Laktašić

Parameters

∗*a*	first value
∗*b*	second value

Returns

returns result of comparison

5.30.2.2 void AK_print_rel_eq_assoc (struct list_node ∗ *list_rel_eq*)

Function for printing RA expresion struct list_node.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.30.2.3 struct list_node∗ AK_rel_eq_assoc (struct list_node ∗ *list_rel_eq*)

Main function for generating RA expresion according to associativity equivalence rules.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.30.2.4 void AK_rel_eq_assoc_test ( )

Function for testing relational equivalences regarding associativity.

Author

Dino Laktašić.

Returns

No return value

5.31 opti/rel_eq_assoc.h File Reference

```
#include "../file/table.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_assoc.h: This graph shows which files directly or indirectly include this file:

Classes

- struct cost_eval_t

  Stucture for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)

Typedefs

- typedef struct cost_eval_t **cost_eval**

Functions

- int AK_compare (const void ∗a, const void ∗b)

  Function for Struct cost_eval comparison.
- struct list_node ∗ AK_rel_eq_assoc (struct list_node ∗list_rel_eq)

  Main function for generating RA expresion according to associativity equivalence rules.
- void AK_print_rel_eq_assoc (struct list_node ∗list_rel_eq)

  Function for printing RA expresion struct list_node.
- void AK_rel_eq_assoc_test ()

  Function for testing relational equivalences regarding associativity.

5.31.1 Detailed Description

Header file that provides data structures for relational equivalences regarding associativity

5.31.2 Function Documentation

5.31.2.1 int AK_compare (const void ∗ *a*, const void ∗ *b*)

Function for Struct cost_eval comparison.

Author

Dino Laktašić

Parameters

∗*a*	first value
∗*b*	second value

Returns

returns result of comparison

5.31.2.2 void AK_print_rel_eq_assoc (struct list_node ∗ *list_rel_eq*)

Function for printing RA expresion struct list_node.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.31.2.3 struct list_node∗ AK_rel_eq_assoc (struct list_node ∗ *list_rel_eq*)

Main function for generating RA expresion according to associativity equivalence rules.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.31.2.4 void AK_rel_eq_assoc_test ( )

Function for testing relational equivalences regarding associativity.

Author

Dino Laktašić.

Returns

No return value

5.32 opti/rel_eq_comut.c File Reference

```
#include "rel_eq_comut.h"
```
Include dependency graph for rel_eq_comut.c:

Functions

- void AK_print_rel_eq_comut (struct list_node *list_rel_eq)

  Function for printing optimized relation equivalence expression list regarding commutativity.
- struct list_node * AK_rel_eq_comut (struct list_node *list_rel_eq)

  Main function for generating RA expresion according to commutativity equivalence rules.
- char * AK_rel_eq_commute_with_theta_join (char *cond, char *tblName)

  Check if selection can commute with theta-join or product.
- void AK_rel_eq_comut_test ()

  relational equivalences regarding commutativity

5.32.1 Detailed Description

Provides functions for relational equivalences regarding commutativity

5.32.2 Function Documentation

5.32.2.1 void AK_print_rel_eq_comut (struct list_node * *list_rel_eq*)

Function for printing optimized relation equivalence expression list regarding commutativity.

Author

Davor Tomala

Parameters

**list_rel_eq*	RA expresion as the struct list_node

5.32.2.2 char* AK_rel_eq_commute_with_theta_join (char * *cond*, char * *tblName*)

Check if selection can commute with theta-join or product.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

Parameters

∗*cond*	condition array that contains condition data
∗*tblName*	name of the table

Returns

pointer to char array that contains new condition for a given table

5.32.2.3 struct list_node∗ AK_rel_eq_comut (struct list_node ∗ *list_rel_eq*)

Main function for generating RA expresion according to commutativity equivalence rules.

Author

Davor Tomala

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.32.2.4 void AK_rel_eq_comut_test ( )

relational equivalences regarding commutativity

Author

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

Returns

No return vlaue

5.33 opti/rel_eq_comut.h File Reference

```
#include "../file/table.h"
#include "./rel_eq_selection.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_comut.h: This graph shows which files directly or indirectly include this file:

Functions

- void AK_print_rel_eq_comut (struct list_node *list_rel_eq)

  Function for printing optimized relation equivalence expression list regarding commutativity.
- struct list_node * AK_rel_eq_comut (struct list_node *list_rel_eq)

  Main function for generating RA expresion according to commutativity equivalence rules.
- char * AK_rel_eq_commute_with_theta_join (char *cond, char *tblName)

  Check if selection can commute with theta-join or product.
- void AK_rel_eq_comut_test ()

  relational equivalences regarding commutativity

5.33.1 Detailed Description

Header file that provides data structures for relational equivalences regarding comutativity

5.33.2 Function Documentation

5.33.2.1 void AK_print_rel_eq_comut (struct list_node * *list_rel_eq*)

Function for printing optimized relation equivalence expression list regarding commutativity.

Author

Davor Tomala

Parameters

**list_rel_eq*	RA expresion as the struct list_node

5.33.2.2 char* AK_rel_eq_commute_with_theta_join (char * *cond,* char * *tblName*)

Check if selection can commute with theta-join or product.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table

2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

Parameters

∗*cond*	condition array that contains condition data
∗*tblName*	name of the table

Returns

pointer to char array that contains new condition for a given table

5.33.2.3 struct list_node∗ AK_rel_eq_comut (struct list_node ∗ *list_rel_eq*)

Main function for generating RA expresion according to commutativity equivalence rules.

Author

Davor Tomala

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.33.2.4 void AK_rel_eq_comut_test ( )

relational equivalences regarding commutativity

Author

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

Returns

No return vlaue

5.34 opti/rel_eq_projection.c File Reference

```
#include "rel_eq_projection.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_projection.c:

Functions

- int AK_rel_eq_is_subset (struct list_node *list_elem_set, struct list_node *list_elem_subset)
  Check if some set of attributes is subset of larger set, used in cascading of the projections.
- int AK_rel_eq_can_commute (struct list_node *list_elem_attribs, struct list_node *list_elem_conds)
  Check if selection uses only attributes retained by the projection before commuting.
- struct list_node * AK_rel_eq_get_attributes (char *tblName)
  Get attributes for a given table and store them to the struct list_node.
- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)
  Filtering and returning only those attributes from list of projection attributes that exist in the given table.
- char * AK_rel_eq_collect_cond_attributes (struct list_node *list_elem)
  Filtering and returning only attributes from selection or theta_join condition.
- char * AK_rel_eq_remove_duplicates (char *attribs)
  Function which removes duplicate attributes from attributes expresion.
- struct list_node * AK_rel_eq_projection (struct list_node *list_rel_eq)
  Main function for generating RA expresion according to projection equivalence rules.
- void AK_print_rel_eq_projection (struct list_node *list_rel_eq)
  Function for printing AK_list to the screen.
- void AK_rel_eq_projection_test ()
  Function for testing rel_eq_selection.

5.34.1 Detailed Description

Provides functions for for relational equivalences in projection

5.34.2 Function Documentation

5.34.2.1 void AK_print_rel_eq_projection (struct list_node * *list_rel_eq*)

Function for printing AK_list to the screen.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the AK_list

Returns

No return value

5.34.2.2 int AK_rel_eq_can_commute (struct list_node * *list_elem_attribs,* struct list_node * *list_elem_conds*)

Check if selection uses only attributes retained by the projection before commuting.

Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break
4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT_SUCCESS, return EXIT_FAILURE

Parameters

list_elem_attribs	list element containing projection data
list_elem_conds	list element containing selection condition data

Returns

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

5.34.2.3 char* AK_rel_eq_collect_cond_attributes (struct list_node * *list_elem*)

Filtering and returning only attributes from selection or theta_join condition.

Author

Dino Laktašić.

Parameters

list_elem	list element that contains selection or theta_join condition data

Returns

only attributes from selection or theta_join condition as the AK_list

5.34.2.4 struct list_node* AK_rel_eq_get_attributes (char * *tblName*)

Get attributes for a given table and store them to the struct list_node.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct list_node
4. For each attribute in table header, insert attribute in struct list_node as new struct list_node element
5. return struct list_node

Parameters

∗*tblName*	name of the table

Returns

struct list_node

5.34.2.5 int AK_rel_eq_is_subset (struct list_node ∗ *list_elem_set,* struct list_node ∗ *list_elem_subset*)

Check if some set of attributes is subset of larger set, used in cascading of the projections.

Author

Dino Laktašić. ============> Optimization plan using Relational Algebra Equivalences <============== Equivalence rule that apply on every equivalent expresion generated by Query optimizer

Rules to implement Rule 1. projection comutes with selection that only uses attributes retained by the projection p[L](s[L1](R)) = s[L1](p[L](R)) Rule 2. only the last in a sequence of projection operations is needed, the others can be omitted. pL1 = p[L1](R) Rule 3a. distribution according to theta join, only if join includes attributes from L1 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (p[L](R2))

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

list_elem_set	first list element containing projection attributes
list_elem_subset	second list element containing projection attributes

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.34.2.6 struct list_node* AK_rel_eq_projection (struct list_node * *list_rel_eq*)

Main function for generating RA expresion according to projection equivalence rules.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the AK_list

Returns

optimised RA expresion as the AK_list

5.34.2.7 char* AK_rel_eq_projection_attributes (char * *attribs,* char * *tblName*)

Filtering and returning only those attributes from list of projection attributes that exist in the given table.

Author

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK_list
4. if exists append attribute to the dynamic atributes char array
5. return pointer to char array with stored attribute/s

Parameters

**attribs*	projection attributes delimited by ";" (ATTR_DELIMITER)
**tblName*	name of the table

Returns

filtered list of projection attributes as the AK_list

5.34.2.8 void AK_rel_eq_projection_test ( )

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.34.2.9 char* AK_rel_eq_remove_duplicates (char * *attribs*)

Function which removes duplicate attributes from attributes expresion.

Author

Dino Laktašić.

Parameters

**attribs*	attributes from which to remove duplicates

Returns

pointer to char array without duplicate attributes

5.35 opti/rel_eq_projection.h File Reference

```
#include "../file/table.h"
#include "../auxi/mempro.h"
```

Include dependency graph for rel_eq_projection.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_rel_eq_is_subset (struct list_node *list_elem_set, struct list_node *list_elem_subset)
  Check if some set of attributes is subset of larger set, used in cascading of the projections.
- int AK_rel_eq_can_commute (struct list_node *list_elem_attribs, struct list_node *list_elem_conds)
  Check if selection uses only attributes retained by the projection before commuting.
- struct list_node * AK_rel_eq_get_attributes (char *tblName)

Get attributes for a given table and store them to the struct list_node.

- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)

  Filtering and returning only those attributes from list of projection attributes that exist in the given table.
- char * AK_rel_eq_collect_cond_attributes (struct list_node *list_elem)

  Filtering and returning only attributes from selection or theta_join condition.
- char * AK_rel_eq_remove_duplicates (char *attribs)

  Function which removes duplicate attributes from attributes expresion.
- struct list_node * AK_rel_eq_projection (struct list_node *list_rel_eq)

  Main function for generating RA expresion according to projection equivalence rules.
- void AK_print_rel_eq_projection (struct list_node *list_rel_eq)

  Function for printing AK_list to the screen.
- void AK_rel_eq_projection_test ()

  Function for testing rel_eq_selection.

5.35.1 Detailed Description

Header file that provides data structures for relational equivalences in projection

5.35.2 Function Documentation

5.35.2.1 void AK_print_rel_eq_projection (struct list_node * *list_rel_eq*)

Function for printing AK_list to the screen.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the AK_list

Returns

No return value

5.35.2.2 int AK_rel_eq_can_commute (struct list_node * *list_elem_attribs,* struct list_node * *list_elem_conds*)

Check if selection uses only attributes retained by the projection before commuting.

Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break

4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT_SUCCESS, return EXIT_FAILURE

Parameters

list_elem_attribs	list element containing projection data
list_elem_conds	list element containing selection condition data

Returns

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

5.35.2.3 char* AK_rel_eq_collect_cond_attributes (struct list_node * *list_elem*)

Filtering and returning only attributes from selection or theta_join condition.

Author

Dino Laktašić.

Parameters

list_elem	list element that contains selection or theta_join condition data

Returns

only attributes from selection or theta_join condition as the AK_list

5.35.2.4 struct list_node* AK_rel_eq_get_attributes (char * *tblName*)

Get attributes for a given table and store them to the struct list_node.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct list_node
4. For each attribute in table header, insert attribute in struct list_node as new struct list_node element
5. return struct list_node

Parameters

**tblName*	name of the table

Returns

struct list_node

5.35.2.5 int AK_rel_eq_is_subset (struct list_node * *list_elem_set,* struct list_node * *list_elem_subset*)

Check if some set of attributes is subset of larger set, used in cascading of the projections.

Author

Dino Laktašić. ============> Optimization plan using Relational Algebra Equivalences <============== Equivalence rule that apply on every equivalent expresion generated by Query optimizer

Rules to implement Rule 1. projection comutes with selection that only uses attributes retained by the projection p[L](s[L1](R)) = s[L1](p[L](R)) Rule 2. only the last in a sequence of projection operations is needed, the others can be omitted. pL1 = p[L1](R) Rule 3a. distribution according to theta join, only if join includes attributes from L1 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (p[L](R2))

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

list_elem_set	first list element containing projection attributes
list_elem_subset	second list element containing projection attributes

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.35.2.6 struct list_node* AK_rel_eq_projection (struct list_node * *list_rel_eq*)

Main function for generating RA expresion according to projection equivalence rules.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the AK_list

Returns

optimised RA expresion as the AK_list

5.35.2.7 char∗ AK_rel_eq_projection_attributes (char ∗ *attribs,* char ∗ *tblName*)

Filtering and returning only those attributes from list of projection attributes that exist in the given table.

Author

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK_list
4. if exists append attribute to the dynamic atributes char array
5. return pointer to char array with stored attribute/s

Parameters

∗*attribs*	projection attributes delimited by ";" (ATTR_DELIMITER)
∗*tblName*	name of the table

Returns

filtered list of projection attributes as the AK_list

5.35.2.8 void AK_rel_eq_projection_test ( )

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.35.2.9 char∗ AK_rel_eq_remove_duplicates (char ∗ *attribs*)

Function which removes duplicate attributes from attributes expresion.

Author

Dino Laktašić.

Parameters

∗attribs	attributes from which to remove duplicates

Returns

pointer to char array without duplicate attributes

5.36 opti/rel_eq_selection.c File Reference

```
#include "rel_eq_selection.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_selection.c:

Functions

- int AK_rel_eq_is_attr_subset (char ∗set, char ∗subset)

  Check if some set of attributes is subset of larger set.
- char ∗ AK_rel_eq_get_atrributes_char (char ∗tblName)

  Get attributes for a given table and store them to the char array.
- char ∗ AK_rel_eq_cond_attributes (char ∗cond)

  Function for filtering and returning attributes from condition.
- int AK_rel_eq_share_attributes (char ∗set, char ∗subset)

  Check if two sets share one or more of it's attributes.
- struct list_node ∗ AK_rel_eq_split_condition (char ∗cond)

  Check if selection can commute with theta-join or product (if working with conditions in infix format use this function insteed - also remember to change code at the other places)
- struct list_node ∗ AK_rel_eq_selection (struct list_node ∗list_rel_eq)

  Main function for generating RA expresion according to selection equivalence rules.
- void AK_print_rel_eq_selection (struct list_node ∗list_rel_eq)

  Function for printing struct list_node to the screen.
- void AK_rel_eq_selection_test ()

  Function for testing rel_eq_selection.

5.36.1 Detailed Description

Provides functions for for relational equivalences in selection

5.36.2 Function Documentation

5.36.2.1 void AK_print_rel_eq_selection (struct list_node ∗ *list_rel_eq*)

Function for printing struct list_node to the screen.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

void

5.36.2.2 char∗ AK_rel_eq_cond_attributes (char ∗ *cond*)

Function for filtering and returning attributes from condition.

Author

Dino Laktašić.

Parameters

∗*cond*	condition array that contains condition data

Returns

pointer to array that contains attributes for a given condition

5.36.2.3 char∗ AK_rel_eq_get_atrributes_char (char ∗ *tblName*)

Get attributes for a given table and store them to the char array.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct list_node
5. For each attribute in table header, insert attribute in the array
6. Delimit each new attribute with ";" (ATTR_DELIMITER)
7. return pointer to char array

Parameters

∗*tblName*	name of the table

Returns

pointer to char array

5.36.2.4 int AK_rel_eq_is_attr_subset (char * *set,* char * *subset*)

Check if some set of attributes is subset of larger set.

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

**set*	set array
**subset*	subset array

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.36.2.5 struct list_node* AK_rel_eq_selection (struct list_node * *list_rel_eq*)

Main function for generating RA expresion according to selection equivalence rules.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.36.2.6 void AK_rel_eq_selection_test ( )

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.36.2.7 int AK_rel_eq_share_attributes (char * *set*, char * *subset*)

Check if two sets share one or more of it's attributes.

Author

Dino Laktašić.

1. If is empty set or subset returns EXIT_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT_SUCCESS
4. else remove unused pointers and return EXIT_FAILURE

Parameters

**set*	first set of attributes delimited by ";" (ATTR_DELIMITER)
**subset*	second set of attributes delimited by ";" (ATTR_DELIMITER)

Returns

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

5.36.2.8 struct list_node* AK_rel_eq_split_condition (char * *cond*)

Check if selection can commute with theta-join or product (if working with conditions in infix format use this function insteed - also remember to change code at the other places)

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition

5. else if token equals to "AND" and id equals to 0 discarge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

Parameters

∗cond	condition array that contains condition data
∗tblName	name of the table

Returns

pointer to char array that contains new condition for a given table

Author

Dino Laktašić. Break conjunctive conditions to individual conditions (currently not used - commented in main AK_rel_eq_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct list_node
2. Remove unused pointers and return the conditions list

Parameters

∗cond	condition expression

Returns

conditions list

5.37 opti/rel_eq_selection.h File Reference

```
#include "../file/table.h"
#include "../auxi/mempro.h"
```

Include dependency graph for rel_eq_selection.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_rel_eq_is_attr_subset (char ∗set, char ∗subset)
  Check if some set of attributes is subset of larger set.
- char ∗ AK_rel_eq_get_atrributes_char (char ∗tblName)
  Get attributes for a given table and store them to the char array.
- char ∗ AK_rel_eq_cond_attributes (char ∗cond)
  Function for filtering and returning attributes from condition.
- int AK_rel_eq_share_attributes (char ∗set, char ∗subset)
  Check if two sets share one or more of it's attributes.

- struct list_node * AK_rel_eq_split_condition (char *cond)

  Check if selection can commute with theta-join or product (if working with conditions in infix format use this function insteed - also remember to change code at the other places)
- struct list_node * AK_rel_eq_selection (struct list_node *list_rel_eq)

  Main function for generating RA expresion according to selection equivalence rules.
- void AK_print_rel_eq_selection (struct list_node *list_rel_eq)

  Function for printing struct list_node to the screen.
- void AK_rel_eq_selection_test ()

  Function for testing rel_eq_selection.

5.37.1 Detailed Description

Header file that provides data structures for relational equivalences in selection

5.37.2 Function Documentation

5.37.2.1 void AK_print_rel_eq_selection (struct list_node * *list_rel_eq*)

Function for printing struct list_node to the screen.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the struct list_node

Returns

void

5.37.2.2 char* AK_rel_eq_cond_attributes (char * *cond*)

Function for filtering and returning attributes from condition.

Author

Dino Laktašić.

Parameters

**cond*	condition array that contains condition data

Returns

pointer to array that contains attributes for a given condition

5.37.2.3 char* AK_rel_eq_get_atrributes_char (char * *tblName*)

Get attributes for a given table and store them to the char array.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct list_node
5. For each attribute in table header, insert attribute in the array
6. Delimit each new attribute with ";" (ATTR_DELIMITER)
7. return pointer to char array

Parameters

**tblName*	name of the table

Returns

pointer to char array

5.37.2.4 int AK_rel_eq_is_attr_subset (char * *set,* char * *subset*)

Check if some set of attributes is subset of larger set.

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

∗*set*	set array
∗*subset*	subset array

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.37.2.5 struct list_node∗ AK_rel_eq_selection (struct list_node ∗ *list_rel_eq*)

Main function for generating RA expresion according to selection equivalence rules.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.37.2.6 void AK_rel_eq_selection_test ( )

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.37.2.7 int AK_rel_eq_share_attributes (char ∗ *set,* char ∗ *subset*)

Check if two sets share one or more of it's attributes.

Author

Dino Laktašić.

1. If is empty set or subset returns EXIT_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT_SUCCESS
4. else remove unused pointers and return EXIT_FAILURE

Parameters

∗*set*	first set of attributes delimited by ";" (ATTR_DELIMITER)
∗*subset*	second set of attributes delimited by ";" (ATTR_DELIMITER)

Returns

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

5.37.2.8 struct list_node∗ AK_rel_eq_split_condition (char ∗ *cond*)

Check if selection can commute with theta-join or product (if working with conditions in infix format use this function insteed - also remember to change code at the other places)

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition
5. else if token equals to "AND" and id equals to 0 discarge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

Parameters

∗*cond*	condition array that contains condition data
∗*tblName*	name of the table

Returns

pointer to char array that contains new condition for a given table

Author

Dino Laktašić. Break conjunctive conditions to individual conditions (currently not used - commented in main AK_rel_eq_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct list_node
2. Remove unused pointers and return the conditions list

Parameters

∗*cond*	condition expression

Returns

conditions list

5.38 rec/archive_log.h File Reference

```
#include "../file/table.h"
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../auxi/mempro.h"
```

Include dependency graph for archive_log.h: This graph shows which files directly or indirectly include this file:

Functions

- void AK_archive_log ()

  Function for making archive log.
- char * AK_get_timestamp ()

  Function that returns the current timestamp.

5.38.1 Detailed Description

Header file that provides data structures for archive logging

5.38.2 Function Documentation

5.38.2.1 void AK_archive_log ( )

Function for making archive log.

Function creates a binary file that stores all commands that failed to execute with a number that shows the size of how many commands failed.

Todo this function takes static filename to store the failed commands, create certain logic that would make the function to use dynamic filename (this is partly implemented inside AK_get_timestamp, but there is no logic that uses the last file when recovering - recovery.c) {link} recovery.c function test

Author

Dražen Bandić, update by Tomislav Turek

Returns

No retun value

5.38.2.2 char* AK_get_timestamp ( )

Function that returns the current timestamp.

This function returns the current timestamp that could be concatenated to a log file in future usages.

Author

Dražen Bandić main logic, replaced by Tomislav Turek

Todo Think about this in the future when creating multiple binary recovery files. Implementation gives the timestamp, but is not used anywhere for now.

Returns

char array in format day.month.year-hour:min:sec.usecu.log

5.39 rec/recovery.c File Reference

`#include "recovery.h"`
Include dependency graph for recovery.c:

Functions

- void AK_recover_archive_log (char *fileName)
  Reads binary file where last commands were saved, and executes them.
- void AK_recovery_insert_row (char *table, char **attributes)
  Inserts a new row in table with attributes.
- char ** AK_recovery_tokenize (char *input, char *delimiter, int valuesOrNot)
  Tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)
- void AK_recover_operation (int sig)
  Function that recovers and executes failed commands.
- void AK_recovery_test ()
  Function for recovery testing.

Variables

- short grandfailure = 0

5.39.1 Detailed Description

Provides recovery functions.

5.39.2 Function Documentation

5.39.2.1 void AK_recover_archive_log (char * *fileName*)

Reads binary file where last commands were saved, and executes them.

Function opens the recovery binary file and executes all commands that were saved inside the redo_log structure

Author

Dražen Bandić, update by Tomislav Turek

Parameters

fileName	- name of the archive log

Returns

no value

5.39.2.2 void AK_recover_operation (int *sig*)

Function that recovers and executes failed commands.

Function is called when SIGINT signal is sent to the system. All commands that are written to rec.bin file are recovered to the designated structure and then executed.

Author

Tomislav Turek

Parameters

sig	required integer parameter for SIGINT handler functions

5.39.2.3 void AK_recovery_insert_row (char * *table,* char ** *attributes*)

Inserts a new row in table with attributes.

Function is given the table name with desired data that should be inserted inside. By using the table name, function retrieves table attributes names and their types which uses afterwards for insert_data_test function to insert data to designated table.

Author

Dražen Bandić, updated by Tomislav Turek

Parameters

table	- table name to insert to
attributes	- attribute to insert

Returns

no value

5.39.2.4 void AK_recovery_test ( )

Function for recovery testing.

Function does nothing while waiting a SIGINT signal (signal represents // doxygen @ for full description ??? system failure). Upon retrieving the signal it calls function AK_recover_operation which starts the recovery by building commands. To comply with the designated structure AK_command_recovery_struct // {link} to struct ??? it writes dummy commands to the file log.log

Author

Tomislav Turek

5.39.2.5 char** AK_recovery_tokenize (char * *input*, char * *delimiter*, int *valuesOrNot*)

Tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)

Author

Dražen Bandić

Parameters

input	- input to tokenize
delimiter	- delimiter
valuesOrNot	- 1 if the input are values, 0 otherwise

Returns

new double pointer structure with tokens

5.39.3 Variable Documentation

5.39.3.1 short grandfailure = 0

this variable flags if system failed

5.40 rec/redo_log.c File Reference

```
#include "redo_log.h"
```

Include dependency graph for redo_log.c:

Functions

- int AK_add_to_redolog (int command, struct list_node *row_root)

  Function adds new element to redolog.
- void AK_printout_redolog ()

  Function prints out the content of redolog memory.
- char * AK_check_attributes (char *attributes)

  Checks if the attribute contains '|', and if it does it replaces it with "\|".

5.40.1 Detailed Description

Provides redolog functions.

5.40.2 Function Documentation

5.40.2.1 int AK_add_to_redolog (int *command,* struct list_node * *row_root*)

Function adds new element to redolog.

Author

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

Returns

EXIT_FAILURE if not allocated memory for ispis, otherwise EXIT_SUCCESS

5.40.2.2 char* AK_check_attributes (char * *attributes*)

Checks if the attribute contains '|', and if it does it replaces it with "\|".

Author

Dražen Bandić

Returns

new attribute

5.40.2.3 void AK_printout_redolog ( )

Function prints out the content of redolog memory.

Author

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

Returns

No return value.

5.41 rel/aggregation.c File Reference

```
#include "aggregation.h"
```

Include dependency graph for aggregation.c:

Functions

- search_result AK_search_unsorted (char *szRelation, search_params *aspParams, int iNum_search_↩ params)

  Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_R↩ ANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.
- int AK_header_size (AK_header *header)

  Function calculates how meany attributes there are in a header with while loop.
- void AK_agg_input_init (AK_agg_input *input)

  Function initializes the input object for aggregation whit init values.
- int AK_agg_input_add (AK_header header, int agg_task, AK_agg_input *input)

  Function adds a header with a task in input object for aggregation.
- int AK_agg_input_add_to_beginning (AK_header header, int agg_task, AK_agg_input *input)

  Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.
- void AK_agg_input_fix (AK_agg_input *input)

  This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TA↩ SK_AVG_COUNT is put on the beginning of input object. After that, AGG_TASK_AVG_SUM is put on the begginig of input object.
- int AK_aggregation (AK_agg_input *input, char *source_table, char *agg_table)

  Function aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK_AVG_↩ SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_↩ values array and results are put in new table.
- void Ak_aggregation_test ()

5.41.1 Detailed Description

Provides functions for aggregation and grouping

5.41.2 Function Documentation

5.41.2.1 int AK_agg_input_add (AK_header *header*, int *agg_task*, AK_agg_input * *input*)

Function adds a header with a task in input object for aggregation.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.41.2.2 int AK_agg_input_add_to_beginning (AK_header *header,* int *agg_task,* AK_agg_input * *input*)

Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.41.2.3 void AK_agg_input_fix (AK_agg_input * *input*)

This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TASK_AVG_COUNT is put on the beginning of input object. After that, AGG_TASK_AVG_SUM is put on the begginig of input object.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.41.2.4 void AK_agg_input_init (AK_agg_input * *input*)

Function initializes the input object for aggregation whit init values.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.41.2.5 int AK_aggregation (AK_agg_input * *input,* char * *source_table,* char * *agg_table*)

Function aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK_AV↩ G_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.

Author

Dejan Frankovic

Parameters

input	input object with list of atributes by which we aggregate and types of aggregations
source_table	- table name for the source table
agg_table	table name for aggregated table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK_insert_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK_insert_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

5.41.2.6 void Ak_aggregation_test ( )

checking results

This variable was added to handle bug described in this file.

5.41.2.7 int AK_header_size (AK_header * *header*)

Function calculates how meany attributes there are in a header with while loop.

Author

Dejan Frankovic

Parameters

header	A header array

Returns

Number of attributes defined in header array

5.41.2.8 search_result AK_search_unsorted (char * *szRelation,* search_params * *aspParams,* int *iNum_search_params*)

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH_RANGE is inclusive. Only one value (or range) per attribute allowed - use search_params.pData_lower for SEARCH_PARTICULAR. Supported types for SEARCH_RANGE: TYPE_INT, TYPE_FLOAT, TYPE_NUMBER, TYPE_DATE, TYPE_DATETIME, TYPE_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

Author

Miroslav Policki

Parameters

szRelation	relation name
aspParams	array of search parameters
iNum_search_params	number of search parameters

Returns

search_result structure defined in filesearch.h. Use AK_deallocate_search_result to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

5.42 rel/aggregation.h File Reference

```
#include "selection.h"
#include "projection.h"
#include "../file/filesearch.h"
#include "../auxi/mempro.h"
```

Include dependency graph for aggregation.h: This graph shows which files directly or indirectly include this file:

Classes

- struct AK_agg_value

  Structure that contains atribute name, date and aggregation task associated.
- struct AK_agg_input

  Structure that contains attributes from table header, tasks for this table and counter value.

Macros

- #define **AGG_TASK_GROUP** 1
- #define **AGG_TASK_COUNT** 2
- #define **AGG_TASK_SUM** 3
- #define **AGG_TASK_MAX** 4
- #define **AGG_TASK_MIN** 5
- #define **AGG_TASK_AVG** 6
- #define **AGG_TASK_AVG_COUNT** 10
- #define **AGG_TASK_AVG_SUM** 11

Functions

- int AK_header_size (AK_header *)

  Function calculates how meany attributes there are in a header with while loop.
- void AK_agg_input_init (AK_agg_input *input)

  Function initializes the input object for aggregation whit init values.
- int AK_agg_input_add (AK_header header, int agg_task, AK_agg_input *input)

  Function adds a header with a task in input object for aggregation.
- int AK_agg_input_add_to_beginning (AK_header header, int agg_task, AK_agg_input *input)

  Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.
- void AK_agg_input_fix (AK_agg_input *input)

  This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TA↩ SK_AVG_COUNT is put on the beginning of input object. After that, AGG_TASK_AVG_SUM is put on the begginig of input object.
- int AK_aggregation (AK_agg_input *input, char *source_table, char *agg_table)

  Function aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK_AVG_↩ SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_↩ values array and results are put in new table.
- void Ak_aggregation_test ()

5.42.1 Detailed Description

Header file that provides data structures for aggregation and grouping

5.42.2 Function Documentation

5.42.2.1 int AK_agg_input_add (AK_header *header,* int *agg_task,* AK_agg_input * *input*)

Function adds a header with a task in input object for aggregation.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.42.2.2 int AK_agg_input_add_to_beginning (AK_header *header,* int *agg_task,* AK_agg_input * *input*)

Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.42.2.3 void AK_agg_input_fix (AK_agg_input * *input*)

This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so,

AGG_TASK_AVG_COUNT is put on the beginning of input object. After that, AGG_TASK_AVG_SUM is put on the begginig of input object.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.42.2.4 void AK_agg_input_init (AK_agg_input * input)

Function initializes the input object for aggregation whit init values.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.42.2.5 int AK_aggregation (AK_agg_input * input, char * source_table, char * agg_table)

Function aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK_AV↩G_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.

Author

Dejan Frankovic

Parameters

input	input object with list of atributes by which we aggregate and types of aggregations
source_table	- table name for the source table
agg_table	table name for aggregated table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK_insert_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK_insert_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

5.42.2.6 void Ak_aggregation_test ( )

checking results

This variable was added to handle bug described in this file.

5.42.2.7 int AK_header_size (AK_header * *header*)

Function calculates how meany attributes there are in a header with while loop.

Author

Dejan Frankovic

Parameters

header	A header array

Returns

Number of attributes defined in header array

5.43 rel/difference.c File Reference

```
#include "difference.h"
```
Include dependency graph for difference.c:

Functions

- int AK_difference (char *srcTable1, char *srcTable2, char *dstTable)

  Function to make difference of the two tables. Table addresses are get through names of tables. Speciallly start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.
- void Ak_op_difference_test ()

  Function for difference operator testing.

5.43.1 Detailed Description

Provides functions for relational difference operation

5.43.2 Function Documentation

5.43.2.1 int AK_difference (char * *srcTable1*, char * *srcTable2*, char * *dstTable*)

Function to make difference of the two tables. Table addresses are get through names of tables. Speciallly start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.43.2.2 void Ak_op_difference_test ( )

Function for difference operator testing.

Author

Dino Laktašić

5.44 rel/difference.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for difference.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_difference (char ∗srcTable1, char ∗srcTable2, char ∗dstTable)

  Function to make difference of the two tables. Table addresses are get through names of tables. Speciallly start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.
- void Ak_op_difference_test ()

  Function for difference operator testing.

5.44.1 Detailed Description

Header file that provides data structures for relational difference operation

5.44.2 Function Documentation

5.44.2.1 int AK_difference (char ∗ *srcTable1,* char ∗ *srcTable2,* char ∗ *dstTable*)

Function to make difference of the two tables. Table addresses are get through names of tables. Speciallly start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.44.2.2 void Ak_op_difference_test ( )

Function for difference operator testing.

Author

Dino Laktašić

5.45 rel/expression_check.c File Reference

```
#include "expression_check.h"
```

Include dependency graph for expression_check.c:

Functions

- int AK_check_arithmetic_statement (struct list_node ∗el, const char ∗op, const char ∗a, const char ∗b)

  Function compares values according to their data type, checks aritmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of el and casts void operands to this type.
- char ∗ AK_replace_wild_card (const char ∗s, char ch, const char ∗repl)

  Function that replaces charachter wildcard (%,_) ch in string s with repl charachters.
- int Ak_check_regex_expression (const char ∗value, const char ∗expression, int sensitive, int checkWildCard)

  Function that evaluates regex expression on given string input.
- int Ak_check_regex_operator_expression (const char ∗value, const char ∗expression)

  Function that evaluates regex expression on given string input.
- int AK_check_if_row_satisfies_expression (struct list_node ∗row_root, struct list_node ∗expr)

  Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function AK_check_arithmetic_statement() is called.
- void **Ak_expression_check_test** ()

5.45.1 Detailed Description

Provides functions for constraint checking used in selection and theta-join

5.45.2 Function Documentation

5.45.2.1 int AK_check_arithmetic_statement (struct list_node ∗ *el,* const char ∗ *op,* const char ∗ *a,* const char ∗ *b*)

Function compares values according to their data type, checks aritmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of el and casts void operands to this type.

Author

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

Parameters

el	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
∗*op*	comparison operator
∗*a*	left operand
∗*b*	right operand

Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

5.45.2.2 int AK_check_if_row_satisfies_expression (struct list_node ∗ *row_root,* struct list_node ∗ *expr*)

Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function AK_check_arithmetic_statement() is called.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

Parameters

row_root	beginning of the row that is to be evaluated
∗*expr*	list with the logical expression in postfix notation

Returns

0 if row does not satisfy, 1 if row satisfies expression

5.45.2.3 int Ak_check_regex_expression (const char ∗ *value,* const char ∗ *expression,* int *sensitive,* int *checkWildCard*)

Function that evaluates regex expression on given string input.

Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression
checkWildCard	replaces SQL wildcard to correesponding POSIX regex charachter
sensitive	case insensitive indicator 1-case sensitive,0- case insensitive

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.45.2.4 int Ak_check_regex_operator_expression (const char * *value*, const char * *expression*)

Function that evaluates regex expression on given string input.

Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.45.2.5 char* AK_replace_wild_card (const char * *s*, char *ch*, const char * *repl*)

Function that replaces charachter wildcard (%,_) ch in string s with repl charachters.

Leon Palaić

Parameters

s	input string
ch	charachter to be replaced

Returns

new sequence of charachters

5.46 rel/expression_check.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
#include <regex.h>
```

Include dependency graph for expression_check.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_check_arithmetic_statement (struct list_node *el, const char *op, const char *a, const char *b)

  Function compares values according to their data type, checks aritmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of el and casts void operands to this type.
- int AK_check_if_row_satisfies_expression (struct list_node *row_root, struct list_node *expr)

  Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function AK_check_arithmetic_statement() is called.

- int Ak_check_regex_expression (const char *value, const char *expression, int sensitive, int checkWildCard)

  Function that evaluates regex expression on given string input.
- int Ak_check_regex_operator_expression (const char *value, const char *expression)

  Function that evaluates regex expression on given string input.
- void **Ak_expression_check_test** ()

5.46.1 Detailed Description

Header file that provides data structures for expression ckecking

5.46.2 Function Documentation

5.46.2.1 int AK_check_arithmetic_statement (struct list_node * *el*, const char * *op*, const char * *a*, const char * *b*)

Function compares values according to their data type, checks aritmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of el and casts void operands to this type.

Author

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

Parameters

el	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
*op	comparison operator
*a	left operand
*b	right operand

Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

5.46.2.2 int AK_check_if_row_satisfies_expression (struct list_node * *row_root*, struct list_node * *expr*)

Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function AK_check_arithmetic_statement() is called.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

Parameters

row_root	beginning of the row that is to be evaluated
*expr	list with the logical expression in postfix notation

Returns

0 if row does not satisfy, 1 if row satisfies expression

5.46.2.3 int Ak_check_regex_expression (const char * *value*, const char * *expression*, int *sensitive*, int *checkWildCard*)

Function that evaluates regex expression on given string input.

Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression
checkWildCard	replaces SQL wildcard to correesponding POSIX regex charachter
sensitive	case insensitive indicator 1-case sensitive,0- case insensitive

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.46.2.4 int Ak_check_regex_operator_expression (const char * *value*, const char * *expression*)

Function that evaluates regex expression on given string input.

Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.47 rel/intersect.c File Reference

`#include "intersect.h"`
Include dependency graph for intersect.c:

Functions

- int AK_intersect (char *srcTable1, char *srcTable2, char *dstTable)

  Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)
- void Ak_op_intersect_test ()

  Function for intersect operator testing.

5.47.1 Detailed Description

Provides functions for relational intersect operation

5.47.2 Function Documentation

5.47.2.1 int AK_intersect (char * *srcTable1,* char * *srcTable2,* char * *dstTable*)

Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.47.2.2 void Ak_op_intersect_test ( )

Function for intersect operator testing.

Author

Dino Laktašić

Returns

No return value

5.48 rel/intersect.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rec/archive_log.h"
#include "../auxi/mempro.h"
```

Include dependency graph for intersect.h: This graph shows which files directly or indirectly include this file:

Classes

- struct intersect_attr

  Structure defines intersect attribute.

Functions

- int AK_intersect (char ∗srcTable1, char ∗srcTable2, char ∗dstTable)

  Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)
- void Ak_op_intersect_test ()

  Function for intersect operator testing.

5.48.1 Detailed Description

Provides data structures for relational intersect operation

5.48.2 Function Documentation

5.48.2.1 int AK_intersect (char ∗ *srcTable1,* char ∗ *srcTable2,* char ∗ *dstTable*)

Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.48.2.2 void Ak_op_intersect_test ( )

Function for intersect operator testing.

Author

Dino Laktašić

Returns

No return value

5.49 rel/nat_join.c File Reference

```
#include "nat_join.h"
```
Include dependency graph for nat_join.c:

Functions

- void AK_create_join_block_header (int table_address1, int table_address2, char *new_table, struct list_node *att)

  Function to make header for the new table and call the function to create the segment.
- void AK_merge_block_join (struct list_node *row_root, struct list_node *row_root_insert, AK_block *temp↩ _block, char *new_table)

  Function searches the second block and when found matches with the first one makes a join and write row to join table.
- void AK_copy_blocks_join (AK_block *tbl1_temp_block, AK_block *tbl2_temp_block, struct list_node *att, char *new_table)

  Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.
- int AK_join (char *srcTable1, char *srcTable2, char *dstTable, struct list_node *att)

  Function to make nat_join betwen two tables on some attributes.
- void AK_op_join_test ()

  Function for natural join testing.

5.49.1 Detailed Description

Provides functions for relational natural join operation

5.49.2 Function Documentation

5.49.2.1 void AK_copy_blocks_join (AK_block * *tbl1_temp_block,* AK_block * *tbl2_temp_block,* struct list_node * *att,* char * *new_table*)

Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
att	attributes on which we make nat_join
new_table	name of the nat_join table

Returns

No return value

5.49.2.2 void AK_create_join_block_header (int *table_address1,* int *table_address2,* char * *new_table,* struct list_node * *att*)

Function to make header for the new table and call the function to create the segment.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

table_address1	address of the block of the first table
table_address2	address of the block of the second table
new_table	name of the join table
att_root	ttributes on which we make nat_join

Returns

No return value

5.49.2.3 int AK_join (char * *srcTable1,* char * *srcTable2,* char * *dstTable,* struct list_node * *att*)

Function to make nat_join betwen two tables on some attributes.

Author

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

Parameters

srcTable1	name of the first table to join
srcTable2	name of the second table to join
att	attributes on which we make nat_join
dstTable	name of the nat_join table

Returns

if success returns EXIT_SUCCESS

5.49.2.4 void AK_merge_block_join (struct list_node * *row_root,* struct list_node * *row_root_insert,* AK_block * *temp_block,* char * *new_table*)

Function searches the second block and when found matches with the first one makes a join and write row to join table.

Author

Matija Novak, updated by Dino Laktašić

Parameters

row_root	- list of values from the first table to be marged with table2
row_root_insert	- list of values from the first table to be inserted into nat_join table
temp_block	- block from the second table to be merged
new_table	- name of the nat_join table

Returns

No return value

5.49.2.5 void AK_op_join_test ( )

Function for natural join testing.

Author

Matija Novak

Returns

No return value

5.50 rel/nat_join.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/projection.h"
#include "../auxi/mempro.h"
```

Include dependency graph for nat_join.h: This graph shows which files directly or indirectly include this file:

Functions

- void AK_create_join_block_header (int table_address1, int table_address2, char ∗new_table, struct list_node ∗att)

  Function to make header for the new table and call the function to create the segment.
- void AK_merge_block_join (struct list_node ∗row_root, struct list_node ∗row_root_insert, AK_block ∗temp↩ _block, char ∗new_table)

  Function searches the second block and when found matches with the first one makes a join and write row to join table.
- void AK_copy_blocks_join (AK_block ∗tbl1_temp_block, AK_block ∗tbl2_temp_block, struct list_node ∗att, char ∗new_table)

  Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.
- int AK_join (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, struct list_node ∗att)

  Function to make nat_join betwen two tables on some attributes.
- void AK_op_join_test ()

  Function for natural join testing.

5.50.1 Detailed Description

Header file that provides data structures for relational natural join operation

5.50.2 Function Documentation

5.50.2.1 void AK_copy_blocks_join (AK_block * *tbl1_temp_block,* AK_block * *tbl2_temp_block,* struct list_node * *att,* char * *new_table*)

Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
att	attributes on which we make nat_join
new_table	name of the nat_join table

Returns

No return value

5.50.2.2 void AK_create_join_block_header (int *table_address1,* int *table_address2,* char * *new_table,* struct list_node * *att*)

Function to make header for the new table and call the function to create the segment.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

table_address1	address of the block of the first table
table_address2	address of the block of the second table
new_table	name of the join table
att_root	ttributes on which we make nat_join

Returns

No return value

5.50.2.3 int AK_join (char * *srcTable1,* char * *srcTable2,* char * *dstTable,* struct list_node * *att*)

Function to make nat_join betwen two tables on some attributes.

Author

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

Parameters

srcTable1	name of the first table to join
srcTable2	name of the second table to join
att	attributes on which we make nat_join
dstTable	name of the nat_join table

Returns

if success returns EXIT_SUCCESS

5.50.2.4 void AK_merge_block_join (struct list_node * *row_root,* struct list_node * *row_root_insert,* AK_block * *temp_block,* char * *new_table*)

Function searches the second block and when found matches with the first one makes a join and write row to join table.

Author

Matija Novak, updated by Dino Laktašić

Parameters

row_root	- list of values from the first table to be marged with table2
row_root_insert	- list of values from the first table to be inserted into nat_join table
temp_block	- block from the second table to be merged
new_table	- name of the nat_join table

Returns

No return value

5.50.2.5 void AK_op_join_test ( )

Function for natural join testing.

Author

Matija Novak

Returns

No return value

5.51 rel/product.c File Reference

`#include "product.h"`
Include dependency graph for product.c:

Functions

- int AK_product (char *srcTable1, char *srcTable2, char *dstTable)
  Function to make product of two tables.
- void AK_op_product_test ()
  Function for product operator testing.

5.51.1 Detailed Description

Provides functions for relational product operation

5.51.2 Function Documentation

5.51.2.1 void AK_op_product_test ( )

Function for product operator testing.

Author

Dino Laktašić

How does this test work? First, it reads all cells from both of the tables, employee and department. After that, it reads all cells from product_test table and compares the data.

Reading data from first two tables (employee and department)

Now reading data from product table and comparing it to the data in first two tables

5.51.2.2 int AK_product (char * *srcTable1*, char * *srcTable2*, char * *dstTable*)

Function to make product of two tables.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

Sorry about indentations, but it is necessary to do it this way, until someone creates function to iterate through table rows (which would be pretty neat, btw.) At this level of code, we have access to rows from the first and from the second table. Then we do it this way: for each row in first table, read row from second table. And concatenate them! Since we have loop hierarchy here, each row from first table will be concatenated with each row from second table. End of story! Let's get back to work... BTW. Please comment your code in the future. It is a lot easier when someone explains to you what he or she was thinking that moment. Write comments in english. Write 'em in croatian. It does not matter! Just explain yourself! And share the idea about comments among others, please. Thank you!

5.52 rel/product.h File Reference

```
#include "../file/table.h"
#include "../file/files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for product.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_product (char ∗srcTable1, char ∗srcTable2, char ∗dstTable)

  Function to make product of two tables.
- void AK_op_product_test ()

  Function for product operator testing.

5.52.1 Detailed Description

Header file that provides data structures for relational product operation

5.52.2 Function Documentation

5.52.2.1 void AK_op_product_test ( )

Function for product operator testing.

Author

Dino Laktašić

How does this test work? First, it reads all cells from both of the tables, employee and department. After that, it reads all cells from product_test table and compares the data.

Reading data from first two tables (employee and department)

Now reading data from product table and comparing it to the data in first two tables

5.52.2.2 int AK_product (char * *srcTable1*, char * *srcTable2*, char * *dstTable*)

Function to make product of two tables.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

Sorry about indentations, but it is necessary to do it this way, until someone creates function to iterate through table rows (which would be pretty neat, btw.) At this level of code, we have access to rows from the first and from the second table. Then we do it this way: for each row in first table, read row from second table. And concatenate them! Since we have loop hierarchy here, each row from first table will be concatenated with each row from second table. End of story! Let's get back to work... BTW. Please comment your code in the future. It is a lot easier when someone explains to you what he or she was thinking that moment. Write comments in english. Write 'em in croatian. It does not matter! Just explain yourself! And share the idea about comments among others, please. Thank you!

5.53 rel/projection.c File Reference

```
#include "projection.h"
```

Include dependency graph for projection.c:

Functions

- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  Temporaly function to create table, and insert entry to the system_relation catalog.
- void AK_create_block_header (int old_block, char *dstTable, struct list_node *att)

  Function to create a new header for the projection table.
- void AK_copy_block_projection (AK_block *old_block, struct list_node *att, char *dstTable, struct list_node *expr)

  Function for copying the data from old table block to the new projection table.
- int AK_projection (char *srcTable, char *dstTable, struct list_node *att, struct list_node *expr)

  Function makes a projection of some table.
- void AK_op_projection_test ()

  Function for projection operator testing.

5.53.1 Detailed Description

Provides functions for relational projection operation

5.53.2 Function Documentation

5.53.2.1 void AK_copy_block_projection (AK_block * *old_block,* struct list_node * *att,* char * *dstTable,* struct list_node * *expr*)

Function for copying the data from old table block to the new projection table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK_list

Parameters

old_block	block from which we copy data
dstTable	name of the new table
att	list of the attributes which should the projeciton table contain No return value

5.53.2.2 void AK_create_block_header (int *old_block,* char * *dstTable,* struct list_node * *att*)

Function to create a new header for the projection table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK_list

Parameters

old_block_add	address of the block from which we copy headers we need
dstTable	name of the new table
att	list of the attributes which should the projeciton table contain

Returns

No return value

5.53.2.3 void AK_op_projection_test ( )

Function for projection operator testing.

Author

Dino Laktašić

Returns

No return value

5.53.2.4 int AK_projection (char * *srcTable,* char * *dstTable,* struct list_node * *att,* struct list_node * *expr*)

Function makes a projection of some table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić, now support cacheing

Parameters

att	- list of atributes on which we make projection
dstTable	table name for projection table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

5.53.2.5 void AK_temp_create_table (char * *table,* AK_header * *header,* int *type_segment*)

Temporaly function to create table, and insert entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.54 rel/projection.h File Reference

```
#include "expression_check.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for projection.h: This graph shows which files directly or indirectly include this file:

Functions

- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  Temporaly function to create table, and insert entry to the system_relation catalog.
- void AK_create_block_header (int old_block, char *dstTable, struct list_node *att)

  Function to create a new header for the projection table.
- void AK_copy_block_projection (AK_block *old_block, struct list_node *att, char *dstTable, struct list_node *expr)

  Function for copying the data from old table block to the new projection table.
- int AK_projection (char *srcTable, char *dstTable, struct list_node *att, struct list_node *expr)

  Function makes a projection of some table.
- void AK_op_projection_test ()

  Function for projection operator testing.

5.54.1 Detailed Description

Header file that provides data structures for relational projection operation

5.54.2 Function Documentation

5.54.2.1 void AK_copy_block_projection (AK_block * *old_block*, struct list_node * *att*, char * *dstTable*, struct list_node * *expr*)

Function for copying the data from old table block to the new projection table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK_list

Parameters

old_block	block from which we copy data
dstTable	name of the new table
att	list of the attributes which should the projeciton table contain No return value

5.54.2.2 void AK_create_block_header (int *old_block,* char * *dstTable,* struct list_node * *att*)

Function to create a new header for the projection table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK_list

Parameters

old_block_add	address of the block from which we copy headers we need
dstTable	name of the new table
att	list of the attributes which should the projeciton table contain

Returns

No return value

5.54.2.3 void AK_op_projection_test ( )

Function for projection operator testing.

Author

Dino Laktašić

Returns

No return value

5.54.2.4 int AK_projection (char * *srcTable,* char * *dstTable,* struct list_node * *att,* struct list_node * *expr*)

Function makes a projection of some table.

Author

Matija Novak, rewrited and optimized by Dino Laktašić, now support cacheing

Parameters

att	- list of atributes on which we make projection
dstTable	table name for projection table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

5.54.2.5 void AK_temp_create_table (char * *table*, AK_header * *header*, int *type_segment*)

Temporaly function to create table, and insert entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.55 rel/selection.c File Reference

```
#include "selection.h"
```

Include dependency graph for selection.c:

Functions

- int AK_selection (char *srcTable, char *dstTable, struct list_node *expr)
  Function which implements selection.
- void AK_op_selection_test ()
  Function for selection operator testing.
- void AK_op_selection_test2 ()
  Function for selection operator testing.
- void AK_op_selection_test_redolog ()
  Function for redolog testing.

5.55.1 Detailed Description

Provides functions for relational selection operation

5.55.2 Function Documentation

5.55.2.1 void AK_op_selection_test ( )

Function for selection operator testing.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

5.55.2.2 void AK_op_selection_test2 ( )

Function for selection operator testing.

Author

Krunoslav Bilić

5.55.2.3 void AK_op_selection_test_redolog ( )

Function for redolog testing.

Author

Krunoslav Bilić

5.55.2.4 int AK_selection (char * *srcTable,* char * *dstTable,* struct list_node * *expr*)

Function which implements selection.

Author

Matija Šestak.

Parameters

**srcTable*	source table name
**dstTable*	destination table name
**expr*	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.56 rel/selection.h File Reference

```
#include "expression_check.h"
#include "../rec/redo_log.h"
#include "../auxi/constants.h"
#include "../auxi/configuration.h"
#include "../file/files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for selection.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_selection (char *srcTable, char *dstTable, struct list_node *expr)
  Function which implements selection.
- void AK_op_selection_test ()
  Function for selection operator testing.
- void AK_op_selection_test2 ()
  Function for selection operator testing.
- void AK_op_selection_test_redolog ()
  Function for redolog testing.

5.56.1 Detailed Description

Header file that provides data structures for relational selection operation

5.56.2 Function Documentation

5.56.2.1 void AK_op_selection_test ( )

Function for selection operator testing.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

5.56.2.2 void AK_op_selection_test2 ( )

Function for selection operator testing.

Author

Krunoslav Bilić

5.56.2.3 void AK_op_selection_test_redolog ( )

Function for redolog testing.

Author

Krunoslav Bilić

5.56.2.4 int AK_selection (char * *srcTable,* char * *dstTable,* struct list_node * *expr*)

Function which implements selection.

Author

Matija Šestak.

Parameters

∗*srcTable*	source table name
∗*dstTable*	destination table name
∗*expr*	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.57 rel/sequence.c File Reference

`#include "sequence.h"`
Include dependency graph for sequence.c:

Functions

- int AK_sequence_add (char ∗name, int start_value, int increment, int max_value, int min_value, int cycle)
  Function for adding sequence.
- int AK_sequence_remove (char ∗name)
  Function for removing sequence.
- int AK_sequence_current_value (char ∗name)
  Function returns the current value of the sequence.
- int AK_sequence_next_value (char ∗name)
  Function returns the next value of the sequence and writes it in a system table as current value.
- int AK_sequence_get_id (char ∗name)
  Function gets sequence id.
- int AK_sequence_rename (char ∗old_name, char ∗new_name)
  Function renames the sequence.
- int AK_sequence_modify (char ∗name, int start_value, int increment, int max_value, int min_value, int cycle)
  Function for modifying sequence.
- void AK_sequence_test ()
  Function for sequences testing.

5.57.1 Detailed Description

Provides functions for sequences

5.57.2 Function Documentation

5.57.2.1 int AK_sequence_add (char ∗ *name*, int *start_value*, int *increment*, int *max_value*, int *min_value*, int *cycle*)

Function for adding sequence.

Author

Boris Kišić

Parameters

name	name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

sequence_id or EXIT_ERROR

5.57.2.2 int AK_sequence_current_value (char * *name*)

Function returns the current value of the sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

current_value or EXIT_ERROR

5.57.2.3 int AK_sequence_get_id (char * *name*)

Function gets sequence id.

Author

Ljubo Barać

Parameters

name	Name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.57.2.4 int AK_sequence_modify (char * *name,* int *start_value,* int *increment,* int *max_value,* int *min_value,* int *cycle*)

Function for modifying sequence.

Author

Boris Kišić fixed by Ljubo Barać

Parameters

name	Name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.57.2.5 int AK_sequence_next_value (char * *name*)

Function returns the next value of the sequence and writes it in a system table as current value.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

next_value or EXIT_ERROR

5.57.2.6 int AK_sequence_remove (char * *name*)

Function for removing sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.57.2.7 int AK_sequence_rename (char * *old_name*, char * *new_name*)

Function renames the sequence.

Author

Boris Kišić

Parameters

old_name	Name of the sequence to be renamed
new_name	New name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.57.2.8 void AK_sequence_test ( )

Function for sequences testing.

Author

Boris Kišić fixed by Ljubo Barać

Returns

No return value

5.58 rel/sequence.h File Reference

```
#include "../file/table.h"
#include "../file/id.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for sequence.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_sequence_add (char *name, int start_value, int increment, int max_value, int min_value, int cycle)
  Function for adding sequence.
- int AK_sequence_remove (char *name)
  Function for removing sequence.
- int AK_sequence_current_value (char *name)
  Function returns the current value of the sequence.
- int AK_sequence_next_value (char *name)
  Function returns the next value of the sequence and writes it in a system table as current value.
- int AK_sequence_rename (char *old_name, char *new_name)
  Function renames the sequence.
- int AK_sequence_modify (char *name, int start_value, int increment, int max_value, int min_value, int cycle)
  Function for modifying sequence.
- int AK_sequence_get_id (char *name)
  Function gets sequence id.
- void AK_sequence_test ()
  Function for sequences testing.

5.58.1 Detailed Description

Header file that provides data structures for sequences

5.58.2 Function Documentation

5.58.2.1 int AK_sequence_add (char * *name*, int *start_value*, int *increment*, int *max_value*, int *min_value*, int *cycle*)

Function for adding sequence.

Author

Boris Kišić

Parameters

name	name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

sequence_id or EXIT_ERROR

5.58.2.2 int AK_sequence_current_value (char * *name*)

Function returns the current value of the sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

current_value or EXIT_ERROR

5.58.2.3 int AK_sequence_get_id (char * *name*)

Function gets sequence id.

Author

Ljubo Barać

Parameters

name	Name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.58.2.4 int AK_sequence_modify (char * *name,* int *start_value,* int *increment,* int *max_value,* int *min_value,* int *cycle*)

Function for modifying sequence.

Author

Boris Kišić fixed by Ljubo Barać

Parameters

name	Name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.58.2.5 int AK_sequence_next_value (char * *name*)

Function returns the next value of the sequence and writes it in a system table as current value.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

next_value or EXIT_ERROR

5.58.2.6 int AK_sequence_remove (char * *name*)

Function for removing sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.58.2.7 int AK_sequence_rename (char * *old_name,* char * *new_name*)

Function renames the sequence.

Author

Boris Kišić

Parameters

old_name	Name of the sequence to be renamed
new_name	New name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.58.2.8 void AK_sequence_test ( )

Function for sequences testing.

Author

Boris Kišić fixed by Ljubo Barać

Returns

No return value

5.59 rel/theta_join.c File Reference

```
#include "theta_join.h"
```

Include dependency graph for theta_join.c:

Functions

- int AK_create_theta_join_header (char ∗srcTable1, char ∗srcTable2, char ∗new_table)

  Function for creating the header of the new table for theta join.
- void AK_check_constraints (AK_block ∗tbl1_temp_block, AK_block ∗tbl2_temp_block, int tbl1_num_att, int tbl2_num_att, struct list_node ∗constraints, char ∗new_table)

  Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.
- int AK_theta_join (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, struct list_node ∗constraints)

  Function for creating a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.
- void AK_op_theta_join_test ()

  Function for testing the theta join.

5.59.1 Detailed Description

Provides functions for relational theta join operation

5.59.2 Function Documentation

5.59.2.1 void AK_check_constraints (AK_block * *tbl1_temp_block*, AK_block * *tbl2_temp_block*, int *tbl1_num_att*, int *tbl2_num_att*, struct list_node * *constraints*, char * *new_table*)

Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

Author

Tomislav Mikulček

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
tbl1_num_att	number of attributes in the first table
tbl2_num_att	number of attributes in the second table
constraints	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
new_table	name of the theta_join table

Returns

No return value

5.59.2.2 int AK_create_theta_join_header (char * *srcTable1*, char * *srcTable2*, char * *new_table*)

Function for creating the header of the new table for theta join.

Author

Tomislav Mikulček

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
new_table	name of the destination table

Returns

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

5.59.2.3 void AK_op_theta_join_test ( )

Function for testing the theta join.

Author

Tomislav Mikulček

Returns

No return value

5.59.2.4 int AK_theta_join (char * *srcTable1*, char * *srcTable2*, char * *dstTable*, struct list_node * *constraints*)

Function for creating a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

Author

Tomislav Mikulček,updated by Nikola Miljancic

Parameters

srcTable1	name of the first table to join
srcTable2	name of the second table to join
constraints	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
dstTable	name of the theta join table

Returns

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

5.60 rel/theta_join.h File Reference

```
#include "expression_check.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for theta_join.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_theta_join (char *srcTable1, char *srcTable2, char *dstTable, struct list_node *constraints)

  Function for creating a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.
- int AK_create_theta_join_header (char *srcTable1, char *srcTable2, char *new_table)

  Function for creating the header of the new table for theta join.
- void AK_check_constraints (AK_block *tbl1_temp_block, AK_block *tbl2_temp_block, int tbl1_num_att, int tbl2_num_att, struct list_node *constraints, char *new_table)

  Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.
- void AK_op_theta_join_test ()

  Function for testing the theta join.

5.60.1 Detailed Description

Header file that provides data structures for theta-join

5.60.2 Function Documentation

5.60.2.1 void AK_check_constraints (AK_block * *tbl1_temp_block*, AK_block * *tbl2_temp_block*, int *tbl1_num_att*, int *tbl2_num_att*, struct list_node * *constraints*, char * *new_table*)

Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

Author

Tomislav Mikulček

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
tbl1_num_att	number of attributes in the first table
tbl2_num_att	number of attributes in the second table
constraints	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
new_table	name of the theta_join table

Returns

No return value

5.60.2.2 int AK_create_theta_join_header (char * *srcTable1*, char * *srcTable2*, char * *new_table*)

Function for creating the header of the new table for theta join.

Author

Tomislav Mikulček

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
new_table	name of the destination table

Returns

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

5.60.2.3 void AK_op_theta_join_test ( )

Function for testing the theta join.

Author

Tomislav Mikulček

Returns

No return value

5.60.2.4 int AK_theta_join (char * *srcTable1,* char * *srcTable2,* char * *dstTable,* struct list_node * *constraints*)

Function for creating a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

Author

Tomislav Mikulček,updated by Nikola Miljancic

Parameters

srcTable1	name of the first table to join
srcTable2	name of the second table to join
constraints	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
dstTable	name of the theta join table

Returns

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

5.61 rel/union.c File Reference

```
#include "union.h"
```

Include dependency graph for union.c:

Functions

- int AK_union (char *srcTable1, char *srcTable2, char *dstTable)

  Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)
- void AK_op_union_test ()

  Function for union operator testing.

5.61.1 Detailed Description

Provides functions for relational union operation

5.61.2 Function Documentation

5.61.2.1 void AK_op_union_test ( )

Function for union operator testing.

Author

Dino Laktašić

Returns

No return value

5.61.2.2 int AK_union (char * *srcTable1*, char * *srcTable2*, char * *dstTable*)

Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.62 rel/union.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for union.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_union (char *srcTable1, char *srcTable2, char *dstTable)

  Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)
- void AK_op_union_test ()

  Function for union operator testing.

5.62.1 Detailed Description

Header file that provides data structures for relational union operation

5.62.2 Function Documentation

5.62.2.1 void AK_op_union_test ( )

Function for union operator testing.

Author

Dino Laktašić

Returns

No return value

5.62.2.2 int AK_union (char * *srcTable1*, char * *srcTable2*, char * *dstTable*)

Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.63 sql/cs/between.c File Reference

`#include "between.h"`
Include dependency graph for between.c:

Functions

- int AK_find_table_address (char *_systemTableName)

  Returns system tables address by name.
- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  Function sets between constraints on particulary attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.
- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  Checks if the given value is between lower and upper bounds of the "between" constraint.
- void Ak_constraint_between_test ()

  Tests the functionality of implemented between constraint.

5.63.1 Detailed Description

Provides functions for between constaint

5.63.2 Function Documentation

5.63.2.1 void Ak_constraint_between_test ( )

Tests the functionality of implemented between constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Returns

No return value

5.63.2.2 int AK_find_table_address (char * *_systemTableName*)

Returns system tables address by name.

Author

Mislav Jurinić

Parameters

_systemTableName	table name

Returns

int

5.63.2.3 int AK_read_constraint_between (char * *tableName,* char * *newValue,* char * *attNamePar*)

Checks if the given value is between lower and upper bounds of the "between" constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
newValue	value we want to insert
attNamePar	attribute name

Returns

EXIT_SUCCESS or EXIT_ERROR

5.63.2.4 void AK_set_constraint_between (char * *tableName,* char * *constraintName,* char * *attName,* char * *startValue,* char * *endValue*)

Function sets between constraints on particulary attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
constraintName	name of constraint
attName	name of attribute
startValue	initial constraint
endValue	final constraint

Returns

No return value

5.64 sql/cs/between.h File Reference

```
#include "../../mm/memoman.h"
#include "../../file/id.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for between.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_find_table_address (char *_systemTableName)

  Returns system tables address by name.
- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  Function sets between constraints on particulary attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.
- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  Checks if the given value is between lower and upper bounds of the "between" constraint.
- void Ak_constraint_between_test ()

  Tests the functionality of implemented between constraint.

5.64.1 Detailed Description

Header file that provides data structures for between constaint

5.64.2 Function Documentation

5.64.2.1 void Ak_constraint_between_test ( )

Tests the functionality of implemented between constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Returns

No return value

5.64.2.2 int AK_find_table_address (char * *_systemTableName*)

Returns system tables address by name.

Author

Mislav Jurinić

Parameters

_systemTableName	table name

Returns

int

5.64.2.3 int AK_read_constraint_between (char * *tableName,* char * *newValue,* char * *attNamePar*)

Checks if the given value is between lower and upper bounds of the "between" constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
newValue	value we want to insert
attNamePar	attribute name

Returns

EXIT_SUCCESS or EXIT_ERROR

5.64.2.4 void AK_set_constraint_between (char * *tableName,* char * *constraintName,* char * *attName,* char * *startValue,* char * *endValue*)

Function sets between constraints on particulary attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
constraintName	name of constraint
attName	name of attribute
startValue	initial constraint
endValue	final constraint

Returns

No return value

5.65 sql/cs/check_constraint.c File Reference

`#include "check_constraint.h"`
Include dependency graph for check_constraint.c:

Functions

- int condition_passed (char *condition, int type, void *value, void *row_data)

  For given value, checks if it satisfies the "check" constraint.
- int AK_set_check_constraint (char *table_name, char *constraint_name, char *attribute_name, char *condition, int type, void *value)

  Adds a new "check" constraint into the system table.
- int AK_check_constraint (char *table, char *attribute, void *value)

  Verifies if the value we want to insert satisfies the "check" constraint.
- void AK_check_constraint_test ()

  Test function for "check" constraint.

5.65.1 Detailed Description

Check constraint implementation file.

5.65.2 Function Documentation

5.65.2.1 int AK_check_constraint (char * *table*, char * *attribute*, void * *value*)

Verifies if the value we want to insert satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

table	target table name
attribute	target attribute name
value	data we want to insert

Returns

1 - success, 0 - failure

5.65.2.2 void AK_check_constraint_test ( )

Test function for "check" constraint.

Author

Mislav Jurinić

Returns

void

5.65.2.3 int AK_set_check_constraint (char * *table_name,* char * *constraint_name,* char * *attribute_name,* char * *condition,* int *type,* void * *value*)

Adds a new "check" constraint into the system table.

Author

Mislav Jurinić

Parameters

table_name	target table for "check" constraint evaluation
constraint_name	new "check" constraint name that will be visible in the system table
attribute_name	target attribute for "check" constraint evaluation
condition	logical operator ['<', '>', '!=', ...]
type	data type [int, float, varchar, datetime, ...]
value	condition to be set

Returns

1 - success, 0 - failure

5.65.2.4 int condition_passed (char * *condition,* int *type,* void * *value,* void * *row_data*)

For given value, checks if it satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

condition	logical operator ['<', '>', '!=', ...]
type	data type [int, float, varchar, datetime, ...]
value	condition to be set
row_data	data in table

Returns

1 - success, 0 - failure

5.66 sql/cs/check_constraint.h File Reference

```
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../rel/expression_check.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for check_constraint.h: This graph shows which files directly or indirectly include this file:

Functions

- int condition_passed (char *condition, int type, void *value, void *row_data)

  For given value, checks if it satisfies the "check" constraint.
- int AK_set_check_constraint (char *table_name, char *constraint_name, char *attribute_name, char *condition, int type, void *value)

  Adds a new "check" constraint into the system table.
- int AK_check_constraint (char *table, char *attribute, void *value)

  Verifies if the value we want to insert satisfies the "check" constraint.
- void AK_check_constraint_test ()

  Test function for "check" constraint.

5.66.1 Detailed Description

Header file that provides data structures for check constraint

5.66.2 Function Documentation

5.66.2.1 int AK_check_constraint (char * *table*, char * *attribute*, void * *value*)

Verifies if the value we want to insert satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

table	target table name
attribute	target attribute name
value	data we want to insert

Returns

1 - success, 0 - failure

5.66.2.2 void AK_check_constraint_test ( )

Test function for "check" constraint.

Author

Mislav Jurinić

Returns

void

5.66.2.3 int AK_set_check_constraint (char * *table_name,* char * *constraint_name,* char * *attribute_name,* char * *condition,* int *type,* void * *value*)

Adds a new "check" constraint into the system table.

Author

Mislav Jurinić

Parameters

table_name	target table for "check" constraint evaluation
constraint_name	new "check" constraint name that will be visible in the system table
attribute_name	target attribute for "check" constraint evaluation
condition	logical operator ['<', '>', '!=', ...]
type	data type [int, float, varchar, datetime, ...]
value	condition to be set

Returns

1 - success, 0 - failure

5.66.2.4 int condition_passed (char * *condition,* int *type,* void * *value,* void * *row_data*)

For given value, checks if it satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

condition	logical operator ['<', '>', '!=', ...]
type	data type [int, float, varchar, datetime, ...]
value	condition to be set
row_data	data in table

Returns

1 - success, 0 - failure

5.67 sql/cs/constraint_names.c File Reference

`#include "constraint_names.h"`
Include dependency graph for constraint_names.c:

Functions

- int Ak_check_constraint_name (char *constraintName)

  Function checks if constraint name would be unique in database.
- void AK_constraint_names_test ()

  Function tests if constraint name would be unique in database.

5.67.1 Detailed Description

Provides functions for checking if constraint name is unique in database

5.67.2 Function Documentation

5.67.2.1 int Ak_check_constraint_name (char * *constraintName*)

Function checks if constraint name would be unique in database.

Author

Nenad Makar, updated by Mislav Jurinić

Parameters

char	constraintName name which you want to give to constraint which you are trying to create

Returns

EXIT_ERROR or EXIT_SUCCESS

TODO add other constraint names from the catalog Also add them to "constants.h"

5.67.2.2 void AK_constraint_names_test ( )

Function tests if constraint name would be unique in database.

Author

Nenad Makar

Returns

No return value

5.68 sql/cs/constraint_names.h File Reference

```
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for constraint_names.h: This graph shows which files directly or indirectly include this file:

Functions

- int Ak_check_constraint_name (char *constraintName)

  Function checks if constraint name would be unique in database.
- void AK_constraint_names_test ()

  Function tests if constraint name would be unique in database.

5.68.1 Detailed Description

Header file that provides functions and data structures for checking if constraint name is unique in database

5.68.2 Function Documentation

5.68.2.1 int Ak_check_constraint_name (char * *constraintName*)

Function checks if constraint name would be unique in database.

Author

Nenad Makar, updated by Mislav Jurinić

Parameters

char	constraintName name which you want to give to constraint which you are trying to create

Returns

EXIT_ERROR or EXIT_SUCCESS

TODO add other constraint names from the catalog Also add them to "constants.h"

5.68.2.2 void AK_constraint_names_test ( )

Function tests if constraint name would be unique in database.

Author

Nenad Makar

Returns

No return value

5.69 sql/cs/nnull.c File Reference

`#include "nnull.h"`
Include dependency graph for nnull.c:

Functions

- int AK_set_constraint_not_null (char *tableName, char *attName, char *constraintName)
  Function that sets NOT NULL constraint on attribute.
- int AK_read_constraint_not_null (char *tableName, char *attName, char *newValue)
  Function checks if there's violation of NOT NULL constraint.
- void AK_null_test ()
  Function for testing testing NOT NULL constraint.

5.69.1 Detailed Description

Provides functions for not null constraint

5.69.2 Function Documentation

5.69.2.1 void AK_null_test ( )

Function for testing testing NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Returns

No return value

5.69.2.2 int AK_read_constraint_not_null (char * *tableName*, char * *attName*, char * *newValue*)

Function checks if there's violation of NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char∗	attName name of attribute
char∗	newValue new value

Returns

EXIT_ERROR or EXIT_SUCCESS

5.69.2.3 int AK_set_constraint_not_null (char ∗ *tableName,* char ∗ *attName,* char ∗ *constraintName*)

Function that sets NOT NULL constraint on attribute.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char∗	attName name of attribute
char∗	constraintName name of constraint

Returns

EXIT_ERROR or EXIT_SUCCESS

5.70 sql/cs/nnull.h File Reference

```
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
#include "constraint_names.h"
```

Include dependency graph for nnull.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_set_constraint_not_null (char ∗tableName, char ∗attName, char ∗constraintName)

  Function that sets NOT NULL constraint on attribute.
- int AK_read_constraint_not_null (char ∗tableName, char ∗attName, char ∗newValue)

  Function checks if there's violation of NOT NULL constraint.
- void AK_null_test ()

  Function for testing testing NOT NULL constraint.

5.70.1 Detailed Description

Header file that provides data structures for not null constraint

5.70.2 Function Documentation

5.70.2.1 void AK_null_test ( )

Function for testing testing NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Returns

No return value

5.70.2.2 int AK_read_constraint_not_null (char * *tableName,* char * *attName,* char * *newValue*)

Function checks if there's violation of NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

*char**	tableName name of table
*char**	attName name of attribute
*char**	newValue new value

Returns

EXIT_ERROR or EXIT_SUCCESS

5.70.2.3 int AK_set_constraint_not_null (char * *tableName,* char * *attName,* char * *constraintName*)

Function that sets NOT NULL constraint on attribute.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char∗	attName name of attribute
char∗	constraintName name of constraint

Returns

EXIT_ERROR or EXIT_SUCCESS

5.71 sql/cs/reference.c File Reference

`#include "reference.h"`
Include dependency graph for reference.c:

Functions

- int AK_add_reference (char ∗childTable, char ∗childAttNames[ ], char ∗parentTable, char ∗parentAttNames[ ], int attNum, char ∗constraintName, int type)

  Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..
- AK_ref_item AK_get_reference (char ∗tableName, char ∗constraintName)

  Function reads a reference entry from system table.
- int AK_reference_check_attribute (char ∗tableName, char ∗attribute, char ∗value)

  Function checks referential integrity for one attribute.
- int AK_reference_check_if_update_needed (struct list_node ∗lista, int action)

  Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.
- int AK_reference_check_restricion (struct list_node ∗lista, int action)

  Function checks for REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.
- int AK_reference_update (struct list_node ∗lista, int action)

  Function updates child table entries according to ongoing update of parent table entries.
- int AK_reference_check_entry (struct list_node ∗lista)

  Function checks new entry for referential integrity.
- void AK_reference_test ()

  Function for testing referential integrity.

5.71.1 Detailed Description

Provides functions for referential integrity

5.71.2 Function Documentation

5.71.2.1 int AK_add_reference (char ∗ *childTable*, char ∗ *childAttNames[ ]*, char ∗ *parentTable*, char ∗ *parentAttNames[ ]*, int *attNum*, char ∗ *constraintName*, int *type*)

Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..

Author

Dejan Frankovic

Parameters

name	of the child table
array	of child table attribute names (foreign key attributes)
name	of the parent table
array	of parent table attribute names (primary key attributes)
number	of attributes in foreign key
name	of the constraint
type	of the constraint, constants defined in 'reference.h'

Returns

EXIT_SUCCESS

5.71.2.2 AK_ref_item AK_get_reference (char * *tableName,* char * *constraintName*)

Function reads a reference entry from system table.

Author

Dejan Frankovic

Parameters

name	of the table with reference (with foreign key)
name	of the reference constraint

Returns

AK_ref_item object with all neccessary information about the reference

5.71.2.3 int AK_reference_check_attribute (char * *tableName,* char * *attribute,* char * *value*)

Function checks referential integrity for one attribute.

Author

Dejan Frankovic

Parameters

child	table name
attribute	name (foreign key attribute)
value	of the attribute we're checking

Returns

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

5.71.2.4 int AK_reference_check_entry (struct list_node * *lista*)

Function checks new entry for referential integrity.

Author

Dejan Franković

Parameters

list	of elements for insert row

Returns

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

5.71.2.5 int AK_reference_check_if_update_needed (struct list_node * *lista,* int *action*)

Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

Author

Dejan Frankovic

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

5.71.2.6 int AK_reference_check_restricion (struct list_node * *lista,* int *action*)

Function checks for REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE?

Returns

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

5.71.2.7 void AK_reference_test ( )

Function for testing referential integrity.

Author

Dejan Franković

Returns

No return value

5.71.2.8 int AK_reference_update (struct list_node * *lista,* int *action*)

Function updates child table entries according to ongoing update of parent table entries.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS

5.72 sql/cs/reference.h File Reference

```
#include "../../dm/dbman.h"
#include "../../file/table.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for reference.h: This graph shows which files directly or indirectly include this file:

Classes

- struct AK_ref_item

  Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

Macros

- #define REF_TYPE_NONE -1

  Constant declaring none reference type.
- #define REF_TYPE_SET_NULL 1

  Constant declaring set null reference type.
- #define REF_TYPE_NO_ACTION 2

  Constant declaring no action reference type.
- #define **REF_TYPE_CASCADE** 3
- #define REF_TYPE_RESTRICT 4

  Constant declaring restrict reference type.
- #define REF_TYPE_SET_DEFAULT 5

  Constant declaring set default reference type.
- #define MAX_REFERENCE_ATTRIBUTES 10

  Constant declaring maximum number of reference attributes.
- #define MAX_CHILD_CONSTRAINTS 20

  Constant declaring maximum number of child constraints.

Functions

- int AK_add_reference (char ∗childTable, char ∗childAttNames[ ], char ∗parentTable, char ∗parentAttNames[ ], int attNum, char ∗constraintName, int type)

  Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..
- AK_ref_item AK_get_reference (char ∗tableName, char ∗constraintName)

  Function reads a reference entry from system table.
- int AK_reference_check_attribute (char ∗tableName, char ∗attribute, char ∗value)

  Function checks referential integrity for one attribute.
- int AK_reference_check_if_update_needed (struct list_node ∗lista, int action)

  Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.
- int AK_reference_check_restricion (struct list_node ∗lista, int action)

  Function checks for REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.
- int AK_reference_update (struct list_node ∗lista, int action)

  Function updates child table entries according to ongoing update of parent table entries.
- int AK_reference_check_entry (struct list_node ∗lista)

  Function checks new entry for referential integrity.
- void AK_reference_test ()

  Function for testing referential integrity.
- void Ak_Insert_New_Element (int newtype, void ∗data, char ∗table, char ∗attribute_name, struct list_node ∗ElementBefore)

  Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.
- int Ak_insert_row (struct list_node ∗row_root)

Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_DIRTY.

- int AK_selection (char *srcTable, char *dstTable, struct list_node *expr)

  Function which implements selection.
- void Ak_Insert_New_Element_For_Update (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore, int newconstraint)

  Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.
- int Ak_delete_row (struct list_node *row_root)

  Function deletes rows.
- int Ak_update_row (struct list_node *row_root)

  Function updates rows of some table.
- int AK_initialize_new_segment (char *name, int type, AK_header *header)

  Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

5.72.1 Detailed Description

Provides data structures for referential integrity

5.72.2 Macro Definition Documentation

5.72.2.1 #define REF_TYPE_NO_ACTION 2

Constant declaring no action reference type.

Constant declaring cascade reference type.

5.72.3 Function Documentation

5.72.3.1 int AK_add_reference (char * *childTable*, char * *childAttNames[ ]*, char * *parentTable*, char * *parentAttNames[ ]*, int *attNum*, char * *constraintName*, int *type*)

Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..

Author

Dejan Frankovic

Parameters

name	of the child table
array	of child table attribute names (foreign key attributes)
name	of the parent table
array	of parent table attribute names (primary key attributes)
number	of attributes in foreign key
name	of the constraint
type	of the constraint, constants defined in 'reference.h'

Returns

EXIT_SUCCESS

5.72.3.2 int Ak_delete_row (struct list_node * *row_root*)

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row EXIT_SUCCESS if success

5.72.3.3 AK_ref_item AK_get_reference (char * *tableName,* char * *constraintName*)

Function reads a reference entry from system table.

Author

Dejan Frankovic

Parameters

name	of the table with reference (with foreign key)
name	of the reference constraint

Returns

AK_ref_item object with all neccessary information about the reference

5.72.3.4 int AK_initialize_new_segment (char * *name,* int *type,* AK_header * *header*)

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

Parameters

name	segment name
type	segment type
header	pointer to header that should be written to the new extent (all blocks)

Returns

start address of new segment

5.72.3.5 void Ak_Insert_New_Element (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore*)

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	is NEW_VALUE

Returns

No return value

5.72.3.6 void Ak_Insert_New_Element_For_Update (int *newtype,* void * *data,* char * *table,* char * *attribute_name,* struct list_node * *ElementBefore,* int *newconstraint*)

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

newtype	type of the data
data	the data
table	table name
attribute_name	attribute name
element	element after we which insert the new element
constraint	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

Returns

No return value

5.72.3.7 int Ak_insert_row (struct list_node * *row_root*)

Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK_↩ DIRTY.

Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK_free, variable table initialized using memset)

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.72.3.8 int AK_reference_check_attribute (char * *tableName,* char * *attribute,* char * *value*)

Function checks referential integrity for one attribute.

Author

Dejan Frankovic

Parameters

child	table name
attribute	name (foreign key attribute)
value	of the attribute we're checking

Returns

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

5.72.3.9 int AK_reference_check_entry (struct list_node * *lista*)

Function checks new entry for referential integrity.

Author

Dejan Franković

Parameters

list	of elements for insert row

Returns

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

5.72.3.10 int AK_reference_check_if_update_needed (struct list_node * *lista,* int *action*)

Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

Author

Dejan Frankovic

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

5.72.3.11 int AK_reference_check_restricion (struct list_node * *lista,* int *action*)

Function checks for REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE?

Returns

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

5.72.3.12 void AK_reference_test ( )

Function for testing referential integrity.

Author

Dejan Franković

Returns

No return value

5.72.3.13 int AK_reference_update (struct list_node * *lista,* int *action*)

Function updates child table entries according to ongoing update of parent table entries.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS

5.72.3.14 int AK_selection (char * *srcTable,* char * *dstTable,* struct list_node * *expr*)

Function which implements selection.

Author

Matija Šestak.

Parameters

*srcTable	source table name
*dstTable	destination table name
*expr	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.72.3.15 int Ak_update_row (struct list_node * *row_root*)

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.73 sql/cs/unique.c File Reference

`#include "unique.h"`
Include dependency graph for unique.c:

Functions

- int Ak_set_constraint_unique (char ∗tableName, char attName[ ], char constraintName[ ])
  Function sets unique constraint on attribute(s)
- int AK_read_constraint_unique (char ∗tableName, char attName[ ], char newValue[ ])
  Function checks if insertion of some value(s) would violate UNIQUE constraint.
- void AK_unique_test ()
  Function for testing UNIQUE constraint.

5.73.1 Detailed Description

Provides functions for unique constraint

5.73.2 Function Documentation

5.73.2.1 int AK_read_constraint_unique (char ∗ *tableName,* char *attName[ ],* char *newValue[ ]*)

Function checks if insertion of some value(s) would violate UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test)
char	newValue[] new value(s), if you want to check combination of values of more attributes seperate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using AK_tuple_to_string(struct list_node ∗tuple) or with sprintf in a similiar way it's used in that function (if value isn't part of a ∗tuple), to concatenate more values in newValue[] use strcat(destination, source) and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

Returns

EXIT_ERROR or EXIT_SUCCESS

5.73.2.2 int Ak_set_constraint_unique (char * *tableName,* char *attName[ ],* char *constraintName[ ]*)

Function sets unique constraint on attribute(s)

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

*char**	tableName name of table
char	attName[] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes seperate their names with constant SEPARATOR (see test)
char	constraintName[] name of constraint

Returns

EXIT_ERROR or EXIT_SUCCESS

5.73.2.3 void AK_unique_test ( )

Function for testing UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Returns

No return value

5.74 sql/cs/unique.h File Reference

```
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
#include "constraint_names.h"
```

Include dependency graph for unique.h: This graph shows which files directly or indirectly include this file:

Functions

- int Ak_set_constraint_unique (char *tableName, char attName[ ], char constraintName[ ])

  Function sets unique constraint on attribute(s)
- int AK_read_constraint_unique (char *tableName, char attName[ ], char newValue[ ])

  Function checks if insertion of some value(s) would violate UNIQUE constraint.
- void AK_unique_test ()

  Function for testing UNIQUE constraint.

5.74.1 Detailed Description

Header file that provides functions and data structures for unique constraint

5.74.2 Function Documentation

5.74.2.1 int AK_read_constraint_unique (char * *tableName,* char *attName[ ],* char *newValue[ ]*)

Function checks if insertion of some value(s) would violate UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

*char**	tableName name of table
char	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test)
char	newValue[] new value(s), if you want to check combination of values of more attributes seperate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using AK_tuple_to_string(struct list_node *tuple) or with sprintf in a similiar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use strcat(destination, source) and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

Returns

EXIT_ERROR or EXIT_SUCCESS

5.74.2.2 int Ak_set_constraint_unique (char * *tableName,* char *attName[ ],* char *constraintName[ ]*)

Function sets unique constraint on attribute(s)

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char	attName[] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes seperate their names with constant SEPARATOR (see test)
char	constraintName[] name of constraint

Returns

EXIT_ERROR or EXIT_SUCCESS

5.74.2.3 void AK_unique_test ( )

Function for testing UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Returns

No return value

5.75 sql/drop.c File Reference

```
#include "drop.h"
```

Include dependency graph for drop.c:

Functions

- void AK_drop (int type, AK_drop_arguments ∗drop_arguments)

  Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.
- void AK_drop_help_function (char ∗tblName, char ∗sys_table)

  Help function for drop command. Delete memory blocks and addresses of table and removes table or index from system table.
- int AK_if_exist (char ∗tblName, char ∗sys_table)

  Help function for check if element(view, function, sequence, user ...) exist in system catalog table.
- void AK_drop_test ()

  Function for testing all DROP functions.

Variables

- char ∗ **system_catalog** [NUM_SYS_TABLES]

5.75.1 Detailed Description

Author

Unknown, Jurica Hlevnjak - drop table bugs fixed, reorganized code structure, system catalog tables drop disabled, drop index added, drop view added, drop sequence added, drop trigger added, drop_function added, drop user added, drop group added, AK_drop_test updated

Provides DROP functions

5.75.2 Function Documentation

5.75.2.1 void AK_drop (int *type,* AK_drop_arguments * *drop_arguments*)

Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.

Author

Unknown, Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

type	drop type
drop_arguments	arguments of DROP command

5.75.2.2 void AK_drop_help_function (char * *tblName,* char * *sys_table*)

Help function for drop command. Delete memory blocks and addresses of table and removes table or index from system table.

Author

unknown, Jurica Hlevnjak - fix bugs and reorganize code in this function

Parameters

tblName	name of table or index
sys_table	name of system catalog table

5.75.2.3 void AK_drop_test ( )

Function for testing all DROP functions.

Author

unknown, Jurica Hlevnjak - added all tests except drop table test, updated by Tomislav Ilisevic

5.75.2.4 int AK_if_exist (char * *tblName,* char * *sys_table*)

Help function for check if element(view, function, sequence, user ...) exist in system catalog table.

Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

tblName	name of table, index view, function, trigger, sequence, user, group or constraint
sys_table	name of system catalog table

Returns

if element exist in system catalog returns 1, if not returns 0

5.75.3 Variable Documentation

5.75.3.1 char* system_catalog[NUM_SYS_TABLES]

Initial value:

```
= {
    "AK_relation",
    "AK_attribute",
    "AK_index",
    "AK_view",
    "AK_sequence",
    "AK_function",
    "AK_function_arguments",
    "AK_trigger",
    "AK_trigger_conditions",
    "AK_db",
    "AK_db_obj",
    "AK_user",
    "AK_group",
    "AK_user_group",
    "AK_user_right",
    "AK_group_right",
    "AK_constraints_between",
    "AK_constraints_not_null",
    AK_CONSTRAINTS_CHECK_CONSTRAINT,
    "AK_constraints_unique",
    "AK_reference"
}
```

5.76 sql/drop.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/sequence.h"
#include "view.h"
#include "trigger.h"
#include "function.h"
#include "privileges.h"
#include "../auxi/mempro.h"
#include "../auxi/constants.h"
```

Include dependency graph for drop.h: This graph shows which files directly or indirectly include this file:

Classes

- struct drop_arguments

Typedefs

- typedef struct drop_arguments **AK_drop_arguments**

Functions

- void AK_drop (int type, AK_drop_arguments *drop_arguments)

  Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.
- void AK_drop_test ()

  Function for testing all DROP functions.
- int AK_if_exist (char *tblName, char *sys_table)

  Help function for check if element(view, function, sequence, user ...) exist in system catalog table.

5.76.1 Function Documentation

5.76.1.1 void AK_drop (int *type*, AK_drop_arguments * *drop_arguments*)

Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.

Author

Unknown, Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

type	drop type
drop_arguments	arguments of DROP command

5.76.1.2 void AK_drop_test ( )

Function for testing all DROP functions.

Author

unknown, Jurica Hlevnjak - added all tests except drop table test, updated by Tomislav Ilisevic

5.76.1.3 int AK_if_exist (char * *tblName*, char * *sys_table*)

Help function for check if element(view, function, sequence, user ...) exist in system catalog table.

Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

tblName	name of table, index view, function, trigger, sequence, user, group or constraint
sys_table	name of system catalog table

Returns

if element exist in system catalog returns 1, if not returns 0

5.77 sql/function.c File Reference

`#include "function.h"`
Include dependency graph for function.c:

Functions

- int AK_get_function_obj_id (char *function, struct list_node *arguments_list)
  Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).
- int AK_check_function_arguments (int function_id, struct list_node *arguments_list)
  Function that checks whether arguments belong to function.
- int AK_check_function_arguments_type (int function_id, struct list_node *args)
  Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.
- int AK_function_add (char *name, int return_type, struct list_node *arguments_list)
  Function that adds a function to system table.
- int AK_function_arguments_add (int function_id, int arg_number, int arg_type, char *argname)
  Function that adds a function argument to system table.
- int AK_function_remove_by_obj_id (int obj_id)
  Function removes a function by its obj_id.
- int AK_function_arguments_remove_by_obj_id (int *obj_id)
  Function removes function arguments by function id.
- int AK_function_remove_by_name (char *name, struct list_node *arguments_list)
  Function that removes a function from system table by name and arguments.
- int AK_function_rename (char *name, struct list_node *arguments_list, char *new_name)
  Function that changes the function name.
- int AK_function_change_return_type (char *name, struct list_node *arguments_list, int new_return_type)
  Function that changes the return type.
- void AK_function_test ()
  Function for functions testing.

5.77.1 Detailed Description

Provides functions for functions

5.77.2 Function Documentation

5.77.2.1 int AK_check_function_arguments (int *function_id,* struct list_node * *arguments_list*)

Function that checks whether arguments belong to function.

Author

Boris Kišić

Parameters

∗*function_id*	id of the function
∗*arguments_list*	list of arguments

Returns

EXIT_SUCCESS of the function or EXIT_ERROR

5.77.2.2 int AK_check_function_arguments_type (int *function_id,* struct list_node ∗ *args*)

Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.

Author

Jurica Hlevnjak

Parameters

function_id	id of the function
args	function arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.3 int AK_function_add (char ∗ *name,* int *return_type,* struct list_node ∗ *arguments_list*)

Function that adds a function to system table.

Author

Boris Kišić, updated by Tomislav Ilisevic

Parameters

∗*name*	name of the function
∗*return_type*	data type returned from a function - values from 0 to 13 - defined in constants.h
∗*arguments_list*	list of function arguments

Returns

function id or EXIT_ERROR

5.77.2.4 int AK_function_arguments_add (int *function_id,* int *arg_number,* int *arg_type,* char * *argname*)

Function that adds a function argument to system table.

Author

Boris Kišić

Parameters

*function_id	id of the function to which the argument belongs
*arg_number	number of the argument
*arg_type	data type of the argument
*argname	name of the argument

Returns

function argument id or EXIT_ERROR

5.77.2.5 int AK_function_arguments_remove_by_obj_id (int * *obj_id*)

Function removes function arguments by function id.

Author

Boris Kišić

Parameters

obj↩ _id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.6 int AK_function_change_return_type (char * *name,* struct list_node * *arguments_list,* int *new_return_type*)

Function that changes the return type.

Author

Boris Kišić

Parameters

*name	name of the function to be modified
*arguments_list	list of function arguments
*new_return_type	new return type

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.7 int AK_function_remove_by_name (char * *name*, struct list_node * *arguments_list*)

Function that removes a function from system table by name and arguments.

Author

Boris Kišić

Parameters

*name	name of the function
*arguments_list	list of arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.8 int AK_function_remove_by_obj_id (int *obj_id*)

Function removes a function by its obj_id.

Author

Boris Kišić

Parameters

obj_id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.9 int AK_function_rename (char * *name*, struct list_node * *arguments_list*, char * *new_name*)

Function that changes the function name.

Author

Boris Kišić

Parameters

∗*name*	name of the function to be modified
∗*arguments_list*	list of arguments to be modified
∗*new_name*	new name of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.77.2.10 void AK_function_test ( )

Function for functions testing.

Author

Boris Kišić, updated by Tomislav Ilisevic

Returns

No return value

5.77.2.11 int AK_get_function_obj_id (char ∗ *function,* struct list_node ∗ *arguments_list*)

Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).

Author

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic

Parameters

∗*function*	name of the function
∗*arguments_list*	list of arguments

Returns

obj_id of the function or EXIT_ERROR

5.78 sql/function.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for function.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_get_function_obj_id (char *function, struct list_node *arguments_list)

  Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).
- int AK_check_function_arguments (int function_id, struct list_node *arguments_list)

  Function that checks whether arguments belong to function.
- int AK_check_function_arguments_type (int function_id, struct list_node *args)

  Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.
- int AK_function_add (char *name, int return_type, struct list_node *arguments_list)

  Function that adds a function to system table.
- int AK_function_arguments_add (int function_id, int arg_number, int arg_type, char *argname)

  Function that adds a function argument to system table.
- int AK_function_remove_by_obj_id (int obj_id)

  Function removes a function by its obj_id.
- int AK_function_arguments_remove_by_obj_id (int *obj_id)

  Function removes function arguments by function id.
- int AK_function_remove_by_name (char *name, struct list_node *arguments_list)

  Function that removes a function from system table by name and arguments.
- int AK_function_rename (char *name, struct list_node *arguments_list, char *new_name)

  Function that changes the function name.
- int AK_function_change_return_type (char *name, struct list_node *arguments_list, int new_return_type)

  Function that changes the return type.
- void AK_function_test ()

  Function for functions testing.

5.78.1 Detailed Description

Header file that provides data structures for functions

Header file that provides data structures functions

5.78.2 Function Documentation

5.78.2.1 int AK_check_function_arguments (int *function_id,* struct list_node * *arguments_list*)

Function that checks whether arguments belong to function.

Author

Boris Kišić

Parameters

**function_id*	id of the function
**arguments_list*	list of arguments

Returns

EXIT_SUCCESS of the function or EXIT_ERROR

5.78.2.2 int AK_check_function_arguments_type (int *function_id,* struct list_node * *args*)

Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.

Author

Jurica Hlevnjak

Parameters

function_id	id of the function
args	function arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.3 int AK_function_add (char * *name,* int *return_type,* struct list_node * *arguments_list*)

Function that adds a function to system table.

Author

Boris Kišić, updated by Tomislav Ilisevic

Parameters

*name	name of the function
*return_type	data type returned from a function - values from 0 to 13 - defined in constants.h
*arguments_list	list of function arguments

Returns

function id or EXIT_ERROR

5.78.2.4 int AK_function_arguments_add (int *function_id,* int *arg_number,* int *arg_type,* char * *argname*)

Function that adds a function argument to system table.

Author

Boris Kišić

Parameters

∗*function_id*	id of the function to which the argument belongs
∗*arg_number*	number of the argument
∗*arg_type*	data type of the argument
∗*argname*	name of the argument

Returns

function argument id or EXIT_ERROR

5.78.2.5 int AK_function_arguments_remove_by_obj_id (int ∗ *obj_id*)

Function removes function arguments by function id.

Author

Boris Kišić

Parameters

obj_id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.6 int AK_function_change_return_type (char ∗ *name,* struct list_node ∗ *arguments_list,* int *new_return_type*)

Function that changes the return type.

Author

Boris Kišić

Parameters

∗*name*	name of the function to be modified
∗*arguments_list*	list of function arguments
∗*new_return_type*	new return type

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.7 int AK_function_remove_by_name (char * name, struct list_node * arguments_list)

Function that removes a function from system table by name and arguments.

Author

Boris Kišić

Parameters

*name	name of the function
*arguments_list	list of arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.8 int AK_function_remove_by_obj_id (int obj_id)

Function removes a function by its obj_id.

Author

Boris Kišić

Parameters

obj_id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.9 int AK_function_rename (char * name, struct list_node * arguments_list, char * new_name)

Function that changes the function name.

Author

Boris Kišić

Parameters

*name	name of the function to be modified
*arguments_list	list of arguments to be modified
*new_name	new name of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.78.2.10 void AK_function_test ( )

Function for functions testing.

Author

Boris Kišić, updated by Tomislav Ilisevic

Returns

No return value

5.78.2.11 int AK_get_function_obj_id (char * *function*, struct list_node * *arguments_list*)

Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).

Author

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic

Parameters

**function*	name of the function
**arguments_list*	list of arguments

Returns

obj_id of the function or EXIT_ERROR

5.79 sql/privileges.c File Reference

```
#include "privileges.h"
```

Include dependency graph for privileges.c:

Functions

- int AK_user_add (char *username, int *password, int set_id)

  Function which inserts user in table AK_user.
- int AK_user_get_id (char *username)

  Function which gets user id.

- int AK_user_remove_by_name (char *name)
  Function removes the user.
- int AK_user_rename (char *old_name, char *new_name, int *password)
  Function renames the user.
- int AK_group_add (char *name, int set_id)
  Function that AK_group_add.
- int AK_group_get_id (char *name)
  Function that gets id of group with given name.
- int AK_group_remove_by_name (char *name)
  Function removes the group.
- int AK_group_rename (char *old_name, char *new_name)
  Function renames the group.
- int AK_grant_privilege_user (char *username, char *table, char *right)
  Function that grants privilege to user.
- int AK_revoke_privilege_user (char *username, char *table, char *right)
  Function that revokes privilege from user on given table.
- int AK_revoke_all_privileges_user (char *username)
  Function that revokes ALL privileges from user on ALL tables (for DROP user)
- int AK_grant_privilege_group (char *groupname, char *table, char *right)
  Function that grants privilege to given group on given table.
- int AK_revoke_privilege_group (char *groupname, char *table, char *right)
  Function that revokes privilege from group on given table.
- int AK_revoke_all_privileges_group (char *groupname)
  Function that revokes ALL privileges from group on ALL tables (needed for DROP group)
- int AK_add_user_to_group (char *user, char *group)
  Function that puts user in given group.
- int AK_remove_user_from_all_groups (char *user)
  Function removes user from all groups. Used for DROP user.
- int AK_remove_all_users_from_group (char *group)
  Function removes all users from group. Used for DROP group.
- int AK_check_privilege (char *username, char *table, char *privilege)
  Function that checks whether given user has right for given operation on given table.
- int AK_check_user_privilege (char *user)
  Function check if user have any privilege or belong to group. Used in drop user for restriction.
- int AK_check_group_privilege (char *group)
  Function check if group have any privilege or user. Used in drop group for restriction.
- void AK_privileges_test ()
  Function that tests functions above for privileges,.

5.79.1 Detailed Description

Provides functions for privileges

5.79.2 Function Documentation

5.79.2.1 int AK_add_user_to_group (char * *user*, char * *group*)

Function that puts user in given group.

Author

Kristina Takač, updated by Mario Peroković, added verifying the existence of user in the group, updated by Maja Vračan

Parameters

∗*user*	username of user which will be put in group
∗*group*	name of group in which user will be put

Returns

EXIT_SUCCESS or EXIT_ERROR if user is already in the group

5.79.2.2 int AK_check_group_privilege (char ∗ *group*)

Function check if group have any privilege or user. Used in drop group for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec

Parameters

group	name of group

Returns

EXIT_ERROR or EXIT_SUCCESS

5.79.2.3 int AK_check_privilege (char ∗ *username,* char ∗ *table,* char ∗ *privilege*)

Function that checks whether given user has right for given operation on given table.

Author

Kristina Takač.

Parameters

∗*user*	username for which we want check privileges
∗*table*	name of table for which we want to check whether user has right on
∗*privilege*	privilege for which we want to check whether user has right for

Returns

EXIT_SUCCESS if user has right, EXIT_ERROR if user has no right

5.79.2.4 int AK_check_user_privilege (char ∗ *user*)

Function check if user have any privilege or belong to group. Used in drop user for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_ERROR or EXIT_SUCCESS

5.79.2.5 int AK_grant_privilege_group (char * *groupname,* char * *table,* char * *right*)

Function that grants privilege to given group on given table.

Author

Kristina Takač.

Parameters

**groupname*	name of group to which we want to grant privilege
**table*	name of table on which privilege will be granted to user
**right*	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.79.2.6 int AK_grant_privilege_user (char * *username,* char * *table,* char * *right*)

Function that grants privilege to user.

Author

Kristina Takač, updated by Mario Peroković, inserting user id instead of username in AK_user_right

Parameters

**username*	username of user to whom we want to grant privilege
**table*	name of table on which privilege will be granted to user
**right*	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.79.2.7 int AK_group_add (char * *name*, int *set_id*)

Function that AK_group_add.

Author

Kristina Takač, edited by Ljubo Barać

Parameters

*name	name of group to be added
set_id	non default id to be passed

Returns

id of group

5.79.2.8 int AK_group_get_id (char * *name*)

Function that gets id of group with given name.

Author

Kristina Takač.

Parameters

*name	name of group whose id we are looking for

Returns

id of group, otherwise EXIT_ERROR

5.79.2.9 int AK_group_remove_by_name (char * *name*)

Function removes the group.

Author

Ljubo Barać

Parameters

name	Name of the group to be removed

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.10 int AK_group_rename (char * *old_name,* char * *new_name*)

Function renames the group.

Author

Ljubo Barać, update by Lidija Lastavec

Parameters

old_name	Name of the group to be renamed
new_name	New name of the group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.11 void AK_privileges_test ( )

Function that tests functions above for privileges,.

Author

Kristina Takač, updated by Tomislav Ilisevic, updated by Lidija Lastavec

Returns

no return value

5.79.2.12 int AK_remove_all_users_from_group (char * *group*)

Function removes all users from group. Used for DROP group.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

group	name of group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.13 int AK_remove_user_from_all_groups (char * *user*)

Function removes user from all groups. Used for DROP user.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.14 int AK_revoke_all_privileges_group (char * *groupname*)

Function that revokes ALL privileges from group on ALL tables (needed for DROP group)

Author

Jurica Hlevnjak

Parameters

groupname	name of group from which we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.79.2.15 int AK_revoke_all_privileges_user (char * *username*)

Function that revokes ALL privileges from user on ALL tables (for DROP user)

Author

Jurica Hlevnjak

Parameters

username	name of user from whom we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.79.2.16 int AK_revoke_privilege_group (char * *groupname,* char * *table,* char * *right*)

Function that revokes privilege from group on given table.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id

Parameters

*grounamep	name of group which user belongs to
*table	name of table on which privilege will be granted to group
*right	type of privilege which will be granted to group on given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.79.2.17 int AK_revoke_privilege_user (char * *username,* char * *table,* char * *right*)

Function that revokes privilege from user on given table.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user_id in AK_user_right

Parameters

*username	username of user to whom we want to grant privilege
*table	name of table on which privilege will be revoked from user
*right	type of privilege which will be revoked from user on given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.79.2.18 int AK_user_add (char * *username*, int * *password*, int *set_id*)

Function which inserts user in table AK_user.

Author

Kristina Takač.

Parameters

**username*	username of user to be added
password	password of user to be added

Returns

user_id

5.79.2.19 int AK_user_get_id (char * *username*)

Function which gets user id.

Author

Kristina Takač.

Parameters

**username*	username of user whose id we are looking for

Returns

user_id, otherwise EXIT_ERROR

5.79.2.20 int AK_user_remove_by_name (char * *name*)

Function removes the user.

Author

Ljubo Barać

Parameters

name	Name of the user to be removed

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.21 int AK_user_rename (char * *old_name,* char * *new_name,* int * *password*)

Function renames the user.

Author

Ljubo Barać, upadate by Lidija Lastavec

Parameters

old_name	Name of the user to be renamed
new_name	New name of the user
password	Password of the user to be renamed (should be provided)

Returns

EXIT_SUCCESS or EXIT_ERROR

5.80 sql/select.c File Reference

```
#include "select.h"
#include "../mm/memoman.h"
```

Include dependency graph for select.c:

Functions

- int AK_select (char *srcTable, char *destTable, struct list_node *attributes, struct list_node *condition)
  Function that implements SELECT relational operator.
- void AK_select_test ()
  Function for testing the implementation.

5.80.1 Detailed Description

Provides functions for SELECT relational operator

5.80.2 Function Documentation

5.80.2.1 int AK_select (char * *srcTable,* char * *destTable,* struct list_node * *attributes,* struct list_node * *condition*)

Function that implements SELECT relational operator.

Author

Renata Mesaros

Parameters

srcTable	- original table that is used for selection
destTable	- table that contains the result
condition	- condition for selection

Returns

EXIT_SUCCESS if cache result in memory and print table else break

calling the relational operator for filtering according to given condition

help table for the final result

new header for the resulting table

going through the header of the table of subscore making a new header for the final result from the selected ones from the subscore

the ordinal number of the selected attribute

if the attribute number is in the selected list, write it in the resulting table

CACHE RESULT IN MEMORY

5.80.2.2 void AK_select_test ( )

Function for testing the implementation.

Author

Renata Mesaros

list of attributes which will be in the result of selection

list of elements which represent the condition for selection

5.81 sql/trigger.c File Reference

```
#include "trigger.h"
```

Include dependency graph for trigger.c:

Functions

- int AK_trigger_save_conditions (int trigger, struct list_node *condition)

  Saves conditions for a trigger.
- int AK_trigger_add (char *name, char *event, struct list_node *condition, char *table, char *function)

  Function that adds a trigger to system table.
- int AK_trigger_get_id (char *name, char *table)

  Function that gets obj_id of a trigger defined by name and table.
- int AK_trigger_remove_by_name (char *name, char *table)

  Function that removes a trigger from system table by name.
- int AK_trigger_remove_by_obj_id (int obj_id)

  Function removes a trigger by its obj_id.
- int AK_trigger_edit (char *name, char *event, struct list_node *condition, char *table, char *function)

  Function edits information about the trigger in system table. In order to identify the trigger, either obj_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.
- struct list_node * AK_trigger_get_conditions (int trigger)

  Function gets postfix list of conditions for the trigger (compatible with selection)
- int AK_trigger_rename (char *old_name, char *new_name, char *table)

  Function renames the trigger.
- void AK_trigger_test ()

  Function for trigger testing.

5.81.1 Detailed Description

Provides functions for triggers

5.81.2 Function Documentation

5.81.2.1 int AK_trigger_add (char * *name*, char * *event*, struct list_node * *condition*, char * *table*, char * *function*)

Function that adds a trigger to system table.

Author

Unknown

Parameters

name	name of the trigger
event	event that calls the trigger - this should perhaps be an integer with defined constants...
condition	AK_list list of conditions in postfix
table	name of the table trigger is hooked on
function	function that is being called by the trigger

Returns

trigger id or EXIT_ERROR

5.81.2.2 int AK_trigger_edit (char * *name,* char * *event,* struct list_node * *condition,* char * *table,* char * *function*)

Function edits information about the trigger in system table. In order to identify the trigger, either obj_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.

Author

Unknown

Parameters

**name*	name of the trigger (or NULL if using obj_id)
**event*	event of the trigger (or NULL if it isn't changing)
**condition*	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
**table*	name of the connected table (or NULL id using obj_id)
**function*	name of the connected function (or NULL if it isn't changing)

Returns

EXIT_SUCCESS or EXIT_ERROR

5.81.2.3 struct list_node* AK_trigger_get_conditions (int *trigger*)

Function gets postfix list of conditions for the trigger (compatible with selection)

Author

Unknown, updated by Mario Peroković

Parameters

trigger	obj_id of the trigger

Returns

list of conditions for the trigger

5.81.2.4 int AK_trigger_get_id (char * *name,* char * *table*)

Function that gets obj_id of a trigger defined by name and table.

Author

Parameters

∗*name*	name of the trigger
∗*table*	name of the table on which the trigger is hooked

Returns

obj_id of the trigger or EXIT_ERROR

5.81.2.5 int AK_trigger_remove_by_name (char ∗ *name,* char ∗ *table*)

Function that removes a trigger from system table by name.

Author

Unknown

Parameters

∗*name*	name of the trigger
∗*table*	name of the table

Returns

EXIT_SUCCESS or EXIT_ERROR

5.81.2.6 int AK_trigger_remove_by_obj_id (int *obj_id*)

Function removes a trigger by its obj_id.

Author

Unknown

Parameters

obj↩ _id	obj_id of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.81.2.7 int AK_trigger_rename (char ∗ *old_name,* char ∗ *new_name,* char ∗ *table*)

Function renames the trigger.

Author

Ljubo Barać

Parameters

old_name	Name of the trigger to be renamed
new_name	New name of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.81.2.8 int AK_trigger_save_conditions (int *trigger,* struct list_node * *condition*)

Saves conditions for a trigger.

Author

Unknown, updated by Mario Peroković, check if data is TYPE_INT

Parameters

trigger	obj_id of the trigger in question
**condition*	AK_list list of conditions

Returns

EXIT_SUCCESS or EXIT_ERROR

5.81.2.9 void AK_trigger_test ( )

Function for trigger testing.

Author

Unknown

5.82 sql/trigger.h File Reference

```
#include "../rec/archive_log.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/id.h"
#include "../sql/function.h"
#include "../rel/selection.h"
#include "../auxi/mempro.h"
```

Include dependency graph for trigger.h: This graph shows which files directly or indirectly include this file:

Functions

- int AK_trigger_save_conditions (int trigger, struct list_node *condition)

  Saves conditions for a trigger.
- int AK_trigger_add (char *name, char *event, struct list_node *condition, char *table, char *function)

  Function that adds a trigger to system table.
- int AK_trigger_get_id (char *name, char *table)

  Function that gets obj_id of a trigger defined by name and table.
- int AK_trigger_remove_by_name (char *name, char *table)

  Function that removes a trigger from system table by name.
- int AK_trigger_remove_by_obj_id (int obj_id)

  Function removes a trigger by its obj_id.
- int AK_trigger_edit (char *name, char *event, struct list_node *condition, char *table, char *function)

  Function edits information about the trigger in system table. In order to identify the trigger, either obj_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.
- struct list_node * AK_trigger_get_conditions (int trigger)

  Function gets postfix list of conditions for the trigger (compatible with selection)
- int AK_trigger_rename (char *old_name, char *new_name, char *table)

  Function renames the trigger.
- void AK_trigger_test ()

  Function for trigger testing.

5.82.1 Detailed Description

Header file that provides data structures triggers

5.82.2 Function Documentation

5.82.2.1 int AK_trigger_add (char * *name*, char * *event*, struct list_node * *condition*, char * *table*, char * *function*)

Function that adds a trigger to system table.

Author

Unknown

Parameters

*name	name of the trigger
*event	event that calls the trigger - this should perhaps be an integer with defined constants...
*condition	AK_list list of conditions in postfix
*table	name of the table trigger is hooked on
*function	function that is being called by the trigger

Returns

trigger id or EXIT_ERROR

5.82.2.2 int AK_trigger_edit (char * *name,* char * *event,* struct list_node * *condition,* char * *table,* char * *function*)

Function edits information about the trigger in system table. In order to identify the trigger, either obj_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.

Author

Unknown

Parameters

**name*	name of the trigger (or NULL if using obj_id)
**event*	event of the trigger (or NULL if it isn't changing)
**condition*	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
**table*	name of the connected table (or NULL id using obj_id)
**function*	name of the connected function (or NULL if it isn't changing)

Returns

EXIT_SUCCESS or EXIT_ERROR

5.82.2.3 struct list_node* AK_trigger_get_conditions (int *trigger*)

Function gets postfix list of conditions for the trigger (compatible with selection)

Author

Unknown, updated by Mario Peroković

Parameters

trigger	obj_id of the trigger

Returns

list of conditions for the trigger

5.82.2.4 int AK_trigger_get_id (char * *name,* char * *table*)

Function that gets obj_id of a trigger defined by name and table.

Author

Parameters

∗*name*	name of the trigger
∗*table*	name of the table on which the trigger is hooked

Returns

obj_id of the trigger or EXIT_ERROR

5.82.2.5 int AK_trigger_remove_by_name (char ∗ *name,* char ∗ *table*)

Function that removes a trigger from system table by name.

Author

Unknown

Parameters

∗*name*	name of the trigger
∗*table*	name of the table

Returns

EXIT_SUCCESS or EXIT_ERROR

5.82.2.6 int AK_trigger_remove_by_obj_id (int *obj_id*)

Function removes a trigger by its obj_id.

Author

Unknown

Parameters

obj↩ _id	obj_id of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.82.2.7 int AK_trigger_rename (char ∗ *old_name,* char ∗ *new_name,* char ∗ *table*)

Function renames the trigger.

Author

Ljubo Barać

Parameters

old_name	Name of the trigger to be renamed
new_name	New name of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.82.2.8 int AK_trigger_save_conditions (int *trigger,* struct list_node * *condition*)

Saves conditions for a trigger.

Author

Unknown, updated by Mario Peroković, check if data is TYPE_INT

Parameters

trigger	obj_id of the trigger in question
**condition*	AK_list list of conditions

Returns

EXIT_SUCCESS or EXIT_ERROR

5.82.2.9 void AK_trigger_test ( )

Function for trigger testing.

Author

Unknown

5.83 sql/view.c File Reference

```
#include "view.h"
```

Include dependency graph for view.c:

Functions

- int AK_get_view_obj_id (char *name)

  Finds an object's id by its name.
- char * AK_get_view_query (char *name)

  Returns a query by its name.
- char * AK_get_rel_exp (char *name)

  Returns a relation expression by its name param name name of the view.
- int AK_view_add (char *name, char *query, char *rel_exp, int set_id)

  Adds a new view to the view table with the corresponding name and value (view query); set_id is optional, if it's not set, the system will determine the new id automatically.
- int AK_view_remove_by_obj_id (int obj_id)

  Removes the view by its object id.
- int AK_view_remove_by_name (char *name)

  Removes the view by its name by identifying the view's id and passing id to AK_view_remove_by_obj_id.
- int AK_view_rename (char *name, char *new_name)

  Renames a view (based on it's name) from "name" to "new_name".
- int AK_view_change_query (char *name, char *query, char *rel_exp)

  Changes the query for a view (determined by it's name) to "query".
- void AK_view_test ()

  A testing function for view.c functions.

5.83.1 Detailed Description

Provides functions for views

5.83.2 Function Documentation

5.83.2.1 char* AK_get_rel_exp (char * *name*)

Returns a relation expression by its name param name name of the view.

Author

Danko Sačer

Returns

rel_exp string or EXIT_ERROR

5.83.2.2 int AK_get_view_obj_id (char * *name*)

Finds an object's id by its name.

Author

Kresimir Ivkovic

Parameters

name	name of the view

Returns

View's id or EXIT_ERROR

5.83.2.3 char* AK_get_view_query (char * *name*)

Returnes a query by its name.

Author

Danko Sačer

Parameters

name	name of the view

Returns

query string or EXIT_ERROR

5.83.2.4 int AK_view_add (char * *name,* char * *query,* char * *rel_exp,* int *set_id*)

Adds a new view to the view table with the corresponding name and value (view query); set_id is optional, if it's not set, the system will determine the new id automatically.

Author

Kresimir Ivkovic

Parameters

name	name og the view
query	query of the view
rel_exp	relation expression of the view
set_id	id of view

Returns

Id of the newly inserted view

5.83.2.5 int AK_view_change_query (char * *name,* char * *query,* char * *rel_exp*)

Changes the query for a view (determined by it's name) to "query".

Author

Kresimir Ivkovic

Parameters

name	of the query
query	new query of the view
rel_exp	relation expression of the view

Returns

error or success

5.83.2.6 int AK_view_remove_by_name (char * *name*)

Removes the view by its name by identifying the view's id and passing id to AK_view_remove_by_obj_id.

Author

Kresimir Ivkovic

Parameters

name	name of the view

Returns

Result of AK_view_remove_by_obj_id or EXIT_ERROR if no id is found

5.83.2.7 int AK_view_remove_by_obj_id (int *obj_id*)

Removes the view by its object id.

Author

Kresimir Ivkovic

Parameters

obj↩ _id	object id of the view

Returns

Result of AK_delete_row for the view (success or error)

5.83.2.8 int AK_view_rename (char * *name*, char * *new_name*)

Renames a view (based on it's name) from "name" to "new_name".

Author

Kresimir Ivkovic

Parameters

name	name of the view
new_name	new name of the view

Returns

error or success

5.83.2.9 void AK_view_test ( )

A testing function for view.c functions.

Author

Kresimir Ivkovic, updated by Lidija Lastavec

5.84 trans/transaction.c File Reference

```
#include "transaction.h"
```

Include dependency graph for transaction.c:

Functions

- int AK_memory_block_hash (int blockMemoryAddress)

  Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.
- AK_transaction_elem_P AK_search_existing_link_for_hook (int blockAddress)

  Searches for a existing entry in hash list of active blocks.
- AK_transaction_elem_P AK_search_empty_link_for_hook (int blockAddress)

  Searches for a empty link for new active block, helper method in case of address collision.
- AK_transaction_elem_P AK_add_hash_entry_list (int blockAddress, int type)

  Adds an element to the doubly linked list.
- int AK_delete_hash_entry_list (int blockAddress)

  Deletes a specific element in the lockTable doubly linked list.
- AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (AK_transaction_elem_P Lockslist, int memoryAddress, pthread_t id)

  Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.
- int AK_delete_lock_entry_list (int blockAddress, pthread_t id)

  Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

- int AK_isLock_waiting (AK_transaction_elem_P lockHolder, int type, pthread_t transactionId, AK_transaction_lock_elem_P lock)

  Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.
- AK_transaction_lock_elem_P AK_add_lock (AK_transaction_elem_P HashList, int type, pthread_t transactionId)

  Adds an element to the locks doubly linked list.
- AK_transaction_lock_elem_P AK_create_lock (int blockAddress, int type, pthread_t transactionId)

  Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.
- int AK_acquire_lock (int memoryAddress, int type, pthread_t transactionId)

  Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.
- void AK_release_locks (AK_memoryAddresses_link addressesTmp, pthread_t transactionId)

  Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .
- int AK_get_memory_blocks (char *tblName, AK_memoryAddresses_link addressList)

  Method that appends all addresses affected by the transaction.
- int AK_execute_commands (command *commandArray, int lengthOfArray)

  Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.
- void * AK_execute_transaction (void *params)

  thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads
- int AK_remove_transaction_thread (pthread_t transaction_thread)

  Function for deleting one of active threads from array of all active transactions threads.
- int AK_create_new_transaction_thread (AK_transaction_data *transaction_data)

  Function for creating new thread. Function also adds thread ID to pthread_t array.
- void AK_transaction_manager (command *commandArray, int lengthOfArray)

  method that receives all the data and gives an id to that data and starts a thread that executes the transaction
- int AK_transaction_register_observer (AK_observable_transaction *observable_transaction, AK_observer *observer)

  Function for registering new observer of AK_observable_transaction type.
- int AK_transaction_unregister_observer (AK_observable_transaction *observable_transaction, AK_observer *observer)

  Function for unregistering observer from AK_observable_transction type.
- void handle_transaction_notify (AK_observer_lock *observer_lock)

  Function for handling AK_observable_transaction notify. Function is associated to some observer instance.
- void AK_on_observable_notify (void *observer, void *observable, AK_ObservableType_Enum type)

  Function for handling notify from some observable type.
- void AK_on_transaction_end (pthread_t transaction_thread)

  Function for handling event when some transaction is finished.
- void AK_on_all_transactions_end ()

  Function for handling event when all transactions are finished.
- void AK_on_lock_release ()

  Function for handling event when one of lock is released.
- void AK_handle_observable_transaction_action (NoticeType *noticeType)

  Function for handling action which is called from observable_transaction type.
- void AK_lock_released ()

  Function which is called when lock is released.
- void AK_transaction_finished ()

  Function which is called when some transaction is finished.
- void AK_all_transactions_finished ()

  Function which is called when all transactions are finished.

- AK_observable_transaction * AK_init_observable_transaction ()

  Function for initialization of AK_observable_transaction type.
- AK_observer_lock * AK_init_observer_lock ()

  Function for initialization of AK_observer_lock type.
- void **AK_test_Transaction** ()

Variables

- AK_transaction_list **LockTable** [NUMBER_OF_KEYS]
- pthread_mutex_t **accessLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **acquireLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **newTransactionLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **endTransationTestLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_cond_t **cond_lock** = PTHREAD_COND_INITIALIZER
- AK_observable_transaction * **observable_transaction**
- pthread_t **activeThreads** [MAX_ACTIVE_TRANSACTIONS_COUNT]
- int **activeTransactionsCount** = 0
- int **transactionsCount** = 0

5.84.1 Detailed Description

Defines functions for transaction execution

5.84.2 Function Documentation

5.84.2.1 int AK_acquire_lock (int *memoryAddress*, int *type*, pthread_t *transactionId*)

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

Author

Frane Jakelić updated by Ivan Pusic

Todo Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

OK or NOT_OK based on the success of the function.

5.84.2.2 AK_transaction_elem_P AK_add_hash_entry_list (int *blockAddress,* int *type*)

Adds an element to the doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
type	of lock issued to the provided memory address.

Returns

pointer to the newly created doubly linked element.

5.84.2.3 AK_transaction_lock_elem_P AK_add_lock (AK_transaction_elem_P *HashList,* int *type,* pthread_t *transactionId*)

Adds an element to the locks doubly linked list.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

pointer to the newly created Locks doubly linked element.

5.84.2.4 void AK_all_transactions_finished ( )

Function which is called when all transactions are finished.

Author

Ivan Pusic

5.84.2.5 AK_transaction_lock_elem_P AK_create_lock (int *blockAddress,* int *type,* pthread_t *transactionId*)

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

pointer to the newly created Locks doubly linked element.

5.84.2.6 int AK_create_new_transaction_thread (AK_transaction_data * *transaction_data*)

Function for creating new thread. Function also adds thread ID to pthread_t array.

Author

Ivan Pusic

Parameters

transaction_data	Data for executing transaction

Returns

Exit status (OK or NOT_OK)

5.84.2.7 int AK_delete_hash_entry_list (int *blockAddress*)

Deletes a specific element in the lockTable doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

integer OK or NOT_OK based on success of finding the specific element in the list.

5.84.2.8 int AK_delete_lock_entry_list (int *blockAddress*, pthread_t *id*)

Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

int OK or NOT_OK based on success of finding the specific element in the list.

5.84.2.9 int AK_execute_commands (command * *commandArray*, int *lengthOfArray*)

Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.

Author

Frane Jakelić updated by Ivan Pusic

Todo Check multithreading, check if it's working correctly

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray
transactionId	associated with the transaction

Returns

ABORT or COMMIT based on the success of the function.

5.84.2.10 void* AK_execute_transaction (void * *params*)

thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads

Author

Frane Jakelić updated by Ivan Pusic

Parameters

data	transmitted to the thread from the main thread

5.84.2.11 int AK_get_memory_blocks (char * *tblName*, AK_memoryAddresses_link *addressList*)

Method that appends all addresses affected by the transaction.

Author

Frane Jakelić

Parameters

addressList	pointer to the linked list where the addresses are stored.
tblName	table name used in the transaction

Returns

OK or NOT_OK based on the success of the function.

5.84.2.12 void AK_handle_observable_transaction_action (NoticeType * *noticeType*)

Function for handling action which is called from observable_transaction type.

Author

Ivan Pusic

Parameters

noticeType	Type of action (event)

5.84.2.13 AK_observable_transaction* AK_init_observable_transaction ( )

Function for initialization of AK_observable_transaction type.

Author

Ivan Pusic

Returns

Pointer to new AK_observable_transaction instance

5.84.2.14 AK_observer_lock* AK_init_observer_lock ( )

Function for initialization of AK_observer_lock type.

Author

Ivan Pusic

Returns

Pointer to new AK_observer_lock instance

5.84.2.15 int AK_isLock_waiting (AK_transaction_elem_P *lockHolder*, int *type*, pthread_t *transactionId*, AK_transaction_lock_elem_P *lock*)

Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

lockHolder	pointer to the hash list entry that is entitled to the specific memory address.
type	of lock issued to the provided memory address.
transaction↩ Id	integer representation of transaction id.
lock	pointer to the lock element that is being tested.

Returns

int PASS_LOCK_QUEUE or WAIT_FOR_UNLOCK based on the rules described inside the function.

5.84.2.16 void AK_lock_released ( )

Function which is called when lock is released.

Author

Ivan Pusic

5.84.2.17 int AK_memory_block_hash (int *blockMemoryAddress*)

Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.

Author

Frane Jakelić

Todo The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

Parameters

blockMemoryAddress	integer representation of memory address, the hash value is calculated from this parameter.

Returns

integer containing the hash value of the passed memory address

5.84.2.18 void AK_on_all_transactions_end ( )

Function for handling event when all transactions are finished.

Author

Ivan Pusic

5.84.2.19 void AK_on_lock_release ( )

Function for handling event when one of lock is released.

Author

Ivan Pusic

5.84.2.20 void AK_on_observable_notify (void * *observer,* void * *observable,* AK_ObservableType_Enum *type*)

Function for handling notify from some observable type.

Author

Ivan Pusic

Parameters

observer	Observer type
observable	Observable type
type	Type of observable who sent some notice

5.84.2.21 void AK_on_transaction_end (pthread_t *transaction_thread*)

Function for handling event when some transaction is finished.

Author

Ivan Pusic

Parameters

transaction_thread	Thread ID of transaction which is finished

5.84.2.22 void AK_release_locks (AK_memoryAddresses_link *addressesTmp*, pthread_t *transactionId*)

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

Author

Frane Jakelić updated by Ivan Pusic

Parameters

adresses	linked list of memory addresses locked by the transaction.
transaction↩ Id	integer representation of transaction id.

5.84.2.23 int AK_remove_transaction_thread (pthread_t *transaction_thread*)

Function for deleting one of active threads from array of all active transactions threads.

Author

Ivan Pusic

Parameters

transaction_thread	Active thread to delete

Returns

Exit status (OK or NOT_OK)

5.84.2.24 AK_transaction_elem_P AK_search_empty_link_for_hook (int *blockAddress*)

Searches for a empty link for new active block, helper method in case of address collision.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to empty location to store new active address

5.84.2.25 AK_transaction_elem_P AK_search_existing_link_for_hook (int *blockAddress*)

Searches for a existing entry in hash list of active blocks.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to the existing hash list entry

5.84.2.26 AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (AK_transaction_elem_P *Lockslist,* int *memoryAddress,* pthread_t *id*)

Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

NULL pointer if the element is not found otherwise it returns a pointer to the found element

5.84.2.27 void AK_transaction_finished ( )

Function which is called when some transaction is finished.

Author

Ivan Pusic

5.84.2.28 void AK_transaction_manager (command * *commandArray*, int *lengthOfArray*)

method that receives all the data and gives an id to that data and starts a thread that executes the transaction

Author

Frane Jakelić updated by Ivan Pusic

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray

5.84.2.29 int AK_transaction_register_observer (AK_observable_transaction * *observable_transaction*, AK_observer * *observer*)

Function for registering new observer of AK_observable_transaction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.84.2.30 int AK_transaction_unregister_observer (AK_observable_transaction * *observable_transaction*, AK_observer * *observer*)

Function for unregistering observer from AK_observable_transction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.84.2.31 void handle_transaction_notify (AK_observer_lock * *observer_lock*)

Function for handling AK_observable_transaction notify. Function is associated to some observer instance.

Author

Ivan Pusic

Parameters

observer_lock	Observer type instance

5.85 trans/transaction.h File Reference

```
#include <pthread.h>
#include "../auxi/constants.h"
#include "../auxi/configuration.h"
#include "../mm/memoman.h"
#include "../sql/command.h"
#include "../auxi/observable.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include <string.h>
#include "../auxi/mempro.h"
```

Include dependency graph for transaction.h: This graph shows which files directly or indirectly include this file:

Classes

- struct observable_transaction_struct
- struct observer_lock

  Structure which defines transaction lock observer type.
- struct transaction_locks_list_elem

  Structure that represents LockTable entry about transaction resource lock.
- struct transaction_list_elem

  Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table.
- struct transaction_list_head

  Structure that represents LockTable entry about doubly linked list of collision in Hash table.

- struct memoryAddresses

  Structure that represents a linked list of locked addresses.
- struct transactionData

  Structure used to transport transaction data to the thread.
- struct threadContainer

  Structure that represents a linked list of threads.

Typedefs

- typedef struct observable_transaction_struct **AK_observable_transaction**
- typedef struct observer_lock **AK_observer_lock**
- typedef struct transactionData **AK_transaction_data**
- typedef struct memoryAddresses **AK_memoryAddresses**
- typedef struct memoryAddresses * **AK_memoryAddresses_link**
- typedef struct transaction_list_head **AK_transaction_list**
- typedef struct transaction_list_elem * **AK_transaction_elem_P**
- typedef struct transaction_list_elem **AK_transaction_elem**
- typedef struct transaction_locks_list_elem * **AK_transaction_lock_elem_P**
- typedef struct transaction_locks_list_elem **AK_transaction_lock_elem**
- typedef struct threadContainer * **AK_thread_elem**
- typedef struct threadContainer **AK_thread_Container**

Enumerations

- enum NoticeType { **AK_LOCK_RELEASED**, **AK_TRANSACTION_FINISHED**, **AK_ALL_TRANSACTION↩ _FINISHED** }

  Enumeration which define notice types for transactions.

Functions

- int AK_memory_block_hash (int)

  Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.
- AK_transaction_elem_P AK_search_existing_link_for_hook (int)

  Searches for a existing entry in hash list of active blocks.
- AK_transaction_elem_P AK_search_empty_link_for_hook (int)

  Searches for a empty link for new active block, helper method in case of address collision.
- AK_transaction_elem_P AK_add_hash_entry_list (int, int)

  Adds an element to the doubly linked list.
- int AK_delete_hash_entry_list (int)

  Deletes a specific element in the lockTable doubly linked list.
- AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (AK_transaction_elem_P, int, pthread_t)

  Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.
- int AK_delete_lock_entry_list (int, pthread_t)

  Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.
- int AK_isLock_waiting (AK_transaction_elem_P, int, pthread_t, AK_transaction_lock_elem_P)

  Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.
- AK_transaction_lock_elem_P AK_add_lock (AK_transaction_elem_P, int, pthread_t)

  Adds an element to the locks doubly linked list.
- AK_transaction_lock_elem_P AK_create_lock (int, int, pthread_t)

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

- int AK_acquire_lock (int, int, pthread_t)

  Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.
- void AK_release_locks (AK_memoryAddresses_link, pthread_t)

  Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .
- int AK_get_memory_blocks (char *, AK_memoryAddresses_link)

  Method that appends all addresses affected by the transaction.
- int AK_execute_commands (command *, int)

  Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.
- void * AK_execute_transaction (void *)

  thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads
- void AK_transaction_manager (command *, int)

  method that receives all the data and gives an id to that data and starts a thread that executes the transaction
- void **AK_test_Transaction** ()
- int AK_create_new_transaction_thread (AK_transaction_data *)

  Function for creating new thread. Function also adds thread ID to pthread_t array.
- int AK_remove_transaction_thread (pthread_t)

  Function for deleting one of active threads from array of all active transactions threads.
- void handle_transaction_notify (AK_observer_lock *)

  Function for handling AK_observable_transaction notify. Function is associated to some observer instance.
- void AK_on_observable_notify (void *, void *, AK_ObservableType_Enum)

  Function for handling notify from some observable type.
- void AK_on_transaction_end (pthread_t)

  Function for handling event when some transaction is finished.
- void AK_on_lock_release ()

  Function for handling event when one of lock is released.
- void AK_on_all_transactions_end ()

  Function for handling event when all transactions are finished.
- void AK_handle_observable_transaction_action (NoticeType *)

  Function for handling action which is called from observable_transaction type.
- void AK_lock_released ()

  Function which is called when lock is released.
- void AK_transaction_finished ()

  Function which is called when some transaction is finished.
- void AK_all_transactions_finished ()

  Function which is called when all transactions are finished.
- int AK_transaction_register_observer (AK_observable_transaction *, AK_observer *)

  Function for registering new observer of AK_observable_transaction type.
- int AK_transaction_unregister_observer (AK_observable_transaction *, AK_observer *)

  Function for unregistering observer from AK_observable_transction type.
- AK_observable_transaction * AK_init_observable_transaction ()

  Function for initialization of AK_observable_transaction type.
- AK_observer_lock * AK_init_observer_lock ()

  Function for initialization of AK_observer_lock type.

5.85.1 Detailed Description

Header file that defines includes and datastructures for the transaction execution

5.85.2 Enumeration Type Documentation

5.85.2.1 enum NoticeType

Enumeration which define notice types for transactions.

Author

Ivan Pusic

5.85.3 Function Documentation

5.85.3.1 int AK_acquire_lock (int *memoryAddress,* int *type,* pthread_t *transactionId*)

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

Author

Frane Jakelić updated by Ivan Pusic

Todo Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

OK or NOT_OK based on the success of the function.

5.85.3.2 AK_transaction_elem_P AK_add_hash_entry_list (int *blockAddress,* int *type*)

Adds an element to the doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
type	of lock issued to the provided memory address.

Returns

pointer to the newly created doubly linked element.

5.85.3.3 AK_transaction_lock_elem_P AK_add_lock (AK_transaction_elem_P *HashList,* int *type,* pthread_t *transactionId*)

Adds an element to the locks doubly linked list.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

pointer to the newly created Locks doubly linked element.

5.85.3.4 void AK_all_transactions_finished ( )

Function which is called when all transactions are finished.

Author

Ivan Pusic

5.85.3.5 AK_transaction_lock_elem_P AK_create_lock (int *blockAddress,* int *type,* pthread_t *transactionId*)

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
type	of lock issued to the provided memory address.
transactionId	integer representation of transaction id.

Returns

pointer to the newly created Locks doubly linked element.

5.85.3.6 int AK_create_new_transaction_thread (AK_transaction_data * *transaction_data*)

Function for creating new thread. Function also adds thread ID to pthread_t array.

Author

Ivan Pusic

Parameters

transaction_data	Data for executing transaction

Returns

Exit status (OK or NOT_OK)

5.85.3.7 int AK_delete_hash_entry_list (int *blockAddress*)

Deletes a specific element in the lockTable doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

integer OK or NOT_OK based on success of finding the specific element in the list.

5.85.3.8 int AK_delete_lock_entry_list (int *blockAddress,* pthread_t *id*)

Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

int OK or NOT_OK based on success of finding the specific element in the list.

5.85.3.9 int AK_execute_commands (command * *commandArray*, int *lengthOfArray*)

Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.

Author

Frane Jakelić updated by Ivan Pusic

Todo Check multithreading, check if it's working correctly

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray
transactionId	associated with the transaction

Returns

ABORT or COMMIT based on the success of the function.

5.85.3.10 void* AK_execute_transaction (void * *params*)

thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads

Author

Frane Jakelić updated by Ivan Pusic

Parameters

data	transmitted to the thread from the main thread

5.85.3.11 int AK_get_memory_blocks (char * *tblName,* AK_memoryAddresses_link *addressList*)

Method that appends all addresses affected by the transaction.

Author

Frane Jakelić

Parameters

addressList	pointer to the linked list where the addresses are stored.
tblName	table name used in the transaction

Returns

OK or NOT_OK based on the success of the function.

5.85.3.12 void AK_handle_observable_transaction_action (NoticeType * *noticeType*)

Function for handling action which is called from observable_transaction type.

Author

Ivan Pusic

Parameters

noticeType	Type of action (event)

5.85.3.13 AK_observable_transaction* AK_init_observable_transaction ( )

Function for initialization of AK_observable_transaction type.

Author

Ivan Pusic

Returns

Pointer to new AK_observable_transaction instance

5.85.3.14 AK_observer_lock* AK_init_observer_lock ( )

Function for initialization of AK_observer_lock type.

Author

Ivan Pusic

Returns

Pointer to new AK_observer_lock instance

5.85.3.15 int AK_isLock_waiting (AK_transaction_elem_P *lockHolder,* int *type,* pthread_t *transactionId,* AK_transaction_lock_elem_P *lock*)

Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

lockHolder	pointer to the hash list entry that is entitled to the specific memory address.
type	of lock issued to the provided memory address.
transaction↩ Id	integer representation of transaction id.
lock	pointer to the lock element that is being tested.

Returns

int PASS_LOCK_QUEUE or WAIT_FOR_UNLOCK based on the rules described inside the function.

5.85.3.16 void AK_lock_released ( )

Function which is called when lock is released.

Author

Ivan Pusic

5.85.3.17 int AK_memory_block_hash (int *blockMemoryAddress*)

Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.

Author

Frane Jakelić

Todo The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

Parameters

blockMemoryAddress	integer representation of memory address, the hash value is calculated from this parameter.

Returns

integer containing the hash value of the passed memory address

5.85.3.18 void AK_on_all_transactions_end ( )

Function for handling event when all transactions are finished.

Author

Ivan Pusic

5.85.3.19 void AK_on_lock_release ( )

Function for handling event when one of lock is released.

Author

Ivan Pusic

5.85.3.20 void AK_on_observable_notify (void * *observer,* void * *observable,* AK_ObservableType_Enum *type*)

Function for handling notify from some observable type.

Author

Ivan Pusic

Parameters

observer	Observer type
observable	Observable type
type	Type of observable who sent some notice

5.85.3.21 void AK_on_transaction_end (pthread_t *transaction_thread*)

Function for handling event when some transaction is finished.

Author

Ivan Pusic

Parameters

transaction_thread	Thread ID of transaction which is finished

5.85.3.22 void AK_release_locks (AK_memoryAddresses_link *addressesTmp*, pthread_t *transactionId*)

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

Author

Frane Jakelić updated by Ivan Pusic

Parameters

adresses	linked list of memory addresses locked by the transaction.
transaction↩ Id	integer representation of transaction id.

5.85.3.23 int AK_remove_transaction_thread (pthread_t *transaction_thread*)

Function for deleting one of active threads from array of all active transactions threads.

Author

Ivan Pusic

Parameters

transaction_thread	Active thread to delete

Returns

Exit status (OK or NOT_OK)

5.85.3.24 AK_transaction_elem_P AK_search_empty_link_for_hook (int *blockAddress*)

Searches for a empty link for new active block, helper method in case of address collision.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to empty location to store new active address

5.85.3.25 AK_transaction_elem_P AK_search_existing_link_for_hook (int *blockAddress*)

Searches for a existing entry in hash list of active blocks.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to the existing hash list entry

5.85.3.26 AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (AK_transaction_elem_P *Lockslist,* int *memoryAddress,* pthread_t *id*)

Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

NULL pointer if the element is not found otherwise it returns a pointer to the found element

5.85.3.27 void AK_transaction_finished ( )

Function which is called when some transaction is finished.

Author

Ivan Pusic

5.85.3.28 void AK_transaction_manager (command * *commandArray*, int *lengthOfArray*)

method that receives all the data and gives an id to that data and starts a thread that executes the transaction

Author

Frane Jakelić updated by Ivan Pusic

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray

5.85.3.29 int AK_transaction_register_observer (AK_observable_transaction * *observable_transaction*, AK_observer * *observer*)

Function for registering new observer of AK_observable_transaction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.85.3.30 int AK_transaction_unregister_observer (AK_observable_transaction * *observable_transaction*, AK_observer * *observer*)

Function for unregistering observer from AK_observable_transction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.85.3.31 void handle_transaction_notify (AK_observer_lock * *observer_lock*)

Function for handling AK_observable_transaction notify. Function is associated to some observer instance.

Author

Ivan Pusic

Parameters

observer_lock	Observer type instance

Index