

Kalashnikov DB  
0.9.2

Generated by Doxygen 1.8.6

Tue Mar 7 2017 16:16:49



# Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Class Documentation</b>	<b>9</b>
4.1	<a href="#">_file_metadata Struct Reference</a> . . . . .	9
4.2	<a href="#">AK_agg_input Struct Reference</a> . . . . .	9
4.2.1	Detailed Description . . . . .	9
4.3	<a href="#">AK_agg_value Struct Reference</a> . . . . .	10
4.3.1	Detailed Description . . . . .	10
4.4	<a href="#">AK_block Struct Reference</a> . . . . .	10
4.4.1	Detailed Description . . . . .	11
4.5	<a href="#">AK_block_activity Struct Reference</a> . . . . .	11
4.5.1	Detailed Description . . . . .	11
4.6	<a href="#">AK_blocktable Struct Reference</a> . . . . .	12
4.7	<a href="#">AK_command_recovery_struct Struct Reference</a> . . . . .	12
4.7.1	Detailed Description . . . . .	12
4.8	<a href="#">AK_command_struct Struct Reference</a> . . . . .	12
4.9	<a href="#">AK_create_table_struct Struct Reference</a> . . . . .	13
4.10	<a href="#">AK_db_cache Struct Reference</a> . . . . .	13
4.10.1	Detailed Description . . . . .	13
4.11	<a href="#">AK_header Struct Reference</a> . . . . .	13
4.11.1	Detailed Description . . . . .	14
4.12	<a href="#">AK_mem_block Struct Reference</a> . . . . .	14
4.12.1	Detailed Description . . . . .	14
4.13	<a href="#">AK_operand Struct Reference</a> . . . . .	15
4.14	<a href="#">AK_query_mem Struct Reference</a> . . . . .	15
4.14.1	Detailed Description . . . . .	15

4.15 AK_query_mem_dict Struct Reference . . . . .	15
4.15.1 Detailed Description . . . . .	16
4.16 AK_query_mem_lib Struct Reference . . . . .	16
4.16.1 Detailed Description . . . . .	16
4.17 AK_query_mem_result Struct Reference . . . . .	16
4.17.1 Detailed Description . . . . .	17
4.18 AK_redo_log Struct Reference . . . . .	17
4.18.1 Detailed Description . . . . .	17
4.19 AK_ref_item Struct Reference . . . . .	17
4.19.1 Detailed Description . . . . .	18
4.20 AK_results Struct Reference . . . . .	18
4.20.1 Detailed Description . . . . .	18
4.21 AK_tuple_dict Struct Reference . . . . .	18
4.21.1 Detailed Description . . . . .	19
4.22 blocktable Struct Reference . . . . .	19
4.22.1 Detailed Description . . . . .	19
4.23 btree_node Struct Reference . . . . .	19
4.24 bucket_elem Struct Reference . . . . .	20
4.24.1 Detailed Description . . . . .	20
4.25 cost_eval_t Struct Reference . . . . .	20
4.25.1 Detailed Description . . . . .	20
4.26 drop_arguments Struct Reference . . . . .	21
4.27 hash_bucket Struct Reference . . . . .	21
4.27.1 Detailed Description . . . . .	21
4.28 hash_info Struct Reference . . . . .	21
4.28.1 Detailed Description . . . . .	22
4.29 intersect_attr Struct Reference . . . . .	22
4.29.1 Detailed Description . . . . .	22
4.30 list_structure_ad Struct Reference . . . . .	22
4.31 list_structure_add Struct Reference . . . . .	23
4.31.1 Detailed Description . . . . .	23
4.32 main_bucket Struct Reference . . . . .	23
4.32.1 Detailed Description . . . . .	23
4.33 memoryAddresses Struct Reference . . . . .	23
4.33.1 Detailed Description . . . . .	24
4.34 observable_transaction Struct Reference . . . . .	24
4.34.1 Detailed Description . . . . .	24
4.35 observable_transaction_struct Struct Reference . . . . .	24
4.36 observer_lock Struct Reference . . . . .	24
4.36.1 Detailed Description . . . . .	25

4.37	<a href="#">root_info Struct Reference</a>	25
4.38	<a href="#">search_params Struct Reference</a>	25
4.38.1	<a href="#">Detailed Description</a>	25
4.39	<a href="#">search_result Struct Reference</a>	26
4.39.1	<a href="#">Detailed Description</a>	26
4.40	<a href="#">struct_add Struct Reference</a>	26
4.40.1	<a href="#">Detailed Description</a>	27
4.41	<a href="#">table_addresses Struct Reference</a>	27
4.41.1	<a href="#">Detailed Description</a>	27
4.42	<a href="#">threadContainer Struct Reference</a>	27
4.42.1	<a href="#">Detailed Description</a>	28
4.43	<a href="#">transaction_list_elem Struct Reference</a>	28
4.43.1	<a href="#">Detailed Description</a>	28
4.44	<a href="#">transaction_list_head Struct Reference</a>	28
4.44.1	<a href="#">Detailed Description</a>	29
4.45	<a href="#">transaction_locks_list_elem Struct Reference</a>	29
4.45.1	<a href="#">Detailed Description</a>	29
4.46	<a href="#">transactionData Struct Reference</a>	29
4.46.1	<a href="#">Detailed Description</a>	30
<b>5</b>	<b>File Documentation</b>	<b>31</b>
5.1	<a href="#">dm/dbman.c File Reference</a>	31
5.1.1	<a href="#">Detailed Description</a>	33
5.1.2	<a href="#">Function Documentation</a>	33
5.1.2.1	<a href="#">AK_allocate_block_activity_modes</a>	33
5.1.2.2	<a href="#">AK_allocate_blocks</a>	33
5.1.2.3	<a href="#">AK_allocationtable_dump</a>	34
5.1.2.4	<a href="#">AK_blocktable_dump</a>	34
5.1.2.5	<a href="#">AK_blocktable_flush</a>	34
5.1.2.6	<a href="#">AK_blocktable_get</a>	34
5.1.2.7	<a href="#">AK_copy_header</a>	34
5.1.2.8	<a href="#">AK_create_header</a>	35
5.1.2.9	<a href="#">AK_delete_block</a>	35
5.1.2.10	<a href="#">AK_delete_extent</a>	36
5.1.2.11	<a href="#">AK_delete_segment</a>	36
5.1.2.12	<a href="#">AK_get_allocation_set</a>	36
5.1.2.13	<a href="#">AK_get_extent</a>	37
5.1.2.14	<a href="#">AK_increase_extent</a>	37
5.1.2.15	<a href="#">AK_init_allocation_table</a>	37
5.1.2.16	<a href="#">AK_init_block</a>	38

5.1.2.17	AK_init_db_file . . . . .	38
5.1.2.18	AK_init_disk_manager . . . . .	38
5.1.2.19	AK_init_system_catalog . . . . .	39
5.1.2.20	AK_init_system_tables_catalog . . . . .	39
5.1.2.21	AK_insert_entry . . . . .	40
5.1.2.22	AK_memset_int . . . . .	40
5.1.2.23	AK_new_extent . . . . .	41
5.1.2.24	AK_new_segment . . . . .	41
5.1.2.25	AK_print_block . . . . .	42
5.1.2.26	AK_read_block . . . . .	42
5.1.2.27	AK_read_block_for_testing . . . . .	42
5.1.2.28	AK_register_system_tables . . . . .	42
5.1.2.29	AK_thread_safe_block_access_test . . . . .	43
5.1.2.30	AK_write_block . . . . .	43
5.1.2.31	AK_write_block_for_testing . . . . .	44
5.1.2.32	fsize . . . . .	44
5.2	dm/dbman.h File Reference . . . . .	44
5.2.1	Detailed Description . . . . .	47
5.2.2	Macro Definition Documentation . . . . .	47
5.2.2.1	AK_ALLOCATION_TABLE_SIZE . . . . .	47
5.2.2.2	CHAR_IN_LINE . . . . .	47
5.2.2.3	MAX_BLOCK_INIT_NUM . . . . .	48
5.2.3	Enumeration Type Documentation . . . . .	48
5.2.3.1	AK_allocation_set_mode . . . . .	48
5.2.4	Function Documentation . . . . .	48
5.2.4.1	AK_allocate_blocks . . . . .	48
5.2.4.2	AK_allocationtable_dump . . . . .	48
5.2.4.3	AK_blocktable_dump . . . . .	48
5.2.4.4	AK_blocktable_flush . . . . .	49
5.2.4.5	AK_blocktable_get . . . . .	49
5.2.4.6	AK_copy_header . . . . .	49
5.2.4.7	AK_create_header . . . . .	49
5.2.4.8	AK_delete_block . . . . .	50
5.2.4.9	AK_delete_extent . . . . .	50
5.2.4.10	AK_delete_segment . . . . .	50
5.2.4.11	AK_get_allocation_set . . . . .	51
5.2.4.12	AK_get_extent . . . . .	51
5.2.4.13	AK_increase_extent . . . . .	52
5.2.4.14	AK_init_allocation_table . . . . .	52
5.2.4.15	AK_init_block . . . . .	52

5.2.4.16	AK_init_db_file	52
5.2.4.17	AK_init_disk_manager	53
5.2.4.18	AK_init_system_catalog	53
5.2.4.19	AK_init_system_tables_catalog	53
5.2.4.20	AK_insert_entry	54
5.2.4.21	AK_memset_int	55
5.2.4.22	AK_new_extent	55
5.2.4.23	AK_new_segment	56
5.2.4.24	AK_print_block	56
5.2.4.25	AK_read_block	57
5.2.4.26	AK_read_block_for_testing	57
5.2.4.27	AK_register_system_tables	57
5.2.4.28	AK_thread_safe_block_access_test	58
5.2.4.29	AK_write_block	58
5.2.4.30	AK_write_block_for_testing	58
5.2.4.31	fsize	59
5.2.5	Variable Documentation	59
5.2.5.1	AK_allocationbit	59
5.2.5.2	db	59
5.2.5.3	db_file_size	59
5.3	file/blobs.c File Reference	59
5.3.1	Detailed Description	60
5.3.2	Function Documentation	60
5.3.2.1	AK_check_folder_blobs	60
5.3.2.2	AK_concat	60
5.3.2.3	AK_folder_exists	61
5.3.2.4	AK_GUID	61
5.3.2.5	AK_lo_export	61
5.3.2.6	AK_lo_import	61
5.3.2.7	AK_lo_test	61
5.3.2.8	AK_lo_unlink	62
5.3.2.9	AK_mkdir	62
5.3.2.10	AK_split_path_file	62
5.4	file/blobs.h File Reference	62
5.4.1	Detailed Description	63
5.4.2	Function Documentation	63
5.4.2.1	AK_check_folder_blobs	63
5.4.2.2	AK_concat	64
5.4.2.3	AK_folder_exists	64
5.4.2.4	AK_GUID	64

5.4.2.5	<a href="#">AK_lo_export</a>	64
5.4.2.6	<a href="#">AK_lo_import</a>	64
5.4.2.7	<a href="#">AK_lo_test</a>	65
5.4.2.8	<a href="#">AK_lo_unlink</a>	65
5.4.2.9	<a href="#">AK_mkdir</a>	65
5.4.2.10	<a href="#">AK_split_path_file</a>	65
5.5	<a href="#">file/fileio.c File Reference</a>	65
5.5.1	<a href="#">Detailed Description</a>	66
5.5.2	<a href="#">Function Documentation</a>	66
5.5.2.1	<a href="#">Ak_delete_row</a>	66
5.5.2.2	<a href="#">Ak_delete_row_by_id</a>	67
5.5.2.3	<a href="#">Ak_delete_row_from_block</a>	67
5.5.2.4	<a href="#">Ak_delete_update_segment</a>	67
5.5.2.5	<a href="#">Ak_Insert_New_Element</a>	67
5.5.2.6	<a href="#">Ak_Insert_New_Element_For_Update</a>	68
5.5.2.7	<a href="#">Ak_insert_row</a>	68
5.5.2.8	<a href="#">Ak_insert_row_to_block</a>	69
5.5.2.9	<a href="#">Ak_update_row</a>	69
5.5.2.10	<a href="#">Ak_update_row_from_block</a>	69
5.6	<a href="#">file/fileio.h File Reference</a>	70
5.6.1	<a href="#">Detailed Description</a>	70
5.6.2	<a href="#">Function Documentation</a>	71
5.6.2.1	<a href="#">Ak_delete_row</a>	71
5.6.2.2	<a href="#">Ak_delete_row_by_id</a>	71
5.6.2.3	<a href="#">Ak_delete_row_from_block</a>	71
5.6.2.4	<a href="#">Ak_delete_update_segment</a>	71
5.6.2.5	<a href="#">Ak_Insert_New_Element</a>	72
5.6.2.6	<a href="#">Ak_Insert_New_Element_For_Update</a>	72
5.6.2.7	<a href="#">Ak_insert_row</a>	73
5.6.2.8	<a href="#">Ak_insert_row_to_block</a>	73
5.6.2.9	<a href="#">Ak_update_row</a>	73
5.6.2.10	<a href="#">Ak_update_row_from_block</a>	74
5.7	<a href="#">file/files.c File Reference</a>	74
5.7.1	<a href="#">Detailed Description</a>	74
5.7.2	<a href="#">Function Documentation</a>	74
5.7.2.1	<a href="#">Ak_files_test</a>	74
5.7.2.2	<a href="#">AK_initialize_new_index_segment</a>	75
5.7.2.3	<a href="#">AK_initialize_new_segment</a>	75
5.8	<a href="#">file/files.h File Reference</a>	75
5.8.1	<a href="#">Detailed Description</a>	76



5.8.2	Function Documentation	76
5.8.2.1	Ak_files_test	76
5.8.2.2	AK_initialize_new_index_segment	76
5.8.2.3	AK_initialize_new_segment	76
5.9	file/filesearch.c File Reference	77
5.9.1	Detailed Description	77
5.9.2	Function Documentation	77
5.9.2.1	AK_deallocate_search_result	77
5.9.2.2	Ak_filesearch_test	77
5.9.2.3	AK_search_unsorted	78
5.10	file/filesearch.h File Reference	78
5.10.1	Detailed Description	79
5.10.2	Function Documentation	79
5.10.2.1	AK_deallocate_search_result	79
5.10.2.2	Ak_filesearch_test	79
5.10.2.3	AK_search_unsorted	80
5.11	file/filesort.h File Reference	80
5.11.1	Detailed Description	81
5.11.2	Function Documentation	81
5.11.2.1	AK_block_sort	81
5.11.2.2	Ak_get_header_number	81
5.11.2.3	Ak_get_num_of_tuples	81
5.11.2.4	Ak_get_total_headers	82
5.12	file/id.c File Reference	82
5.12.1	Detailed Description	82
5.12.2	Function Documentation	82
5.12.2.1	AK_get_id	82
5.12.2.2	AK_get_table_id	82
5.12.2.3	Ak_id_test	83
5.13	file/id.h File Reference	83
5.13.1	Detailed Description	83
5.13.2	Function Documentation	83
5.13.2.1	AK_get_id	83
5.13.2.2	Ak_id_test	84
5.14	file/idx/bitmap.c File Reference	84
5.14.1	Detailed Description	85
5.14.2	Function Documentation	85
5.14.2.1	AK_add_to_bitmap_index	85
5.14.2.2	Ak_bitmap_test	85
5.14.2.3	Ak_create_Index	85

5.14.2.4	<a href="#">AK_create_Index_Table</a>	86
5.14.2.5	<a href="#">AK_delete_bitmap_index</a>	86
5.14.2.6	<a href="#">Ak_get_Attribute</a>	86
5.14.2.7	<a href="#">AK_get_Attribute</a>	87
5.14.2.8	<a href="#">Ak_If_ExistOp</a>	87
5.14.2.9	<a href="#">Ak_print_Att_Test</a>	87
5.14.2.10	<a href="#">Ak_print_Header_Test</a>	87
5.14.2.11	<a href="#">AK_update</a>	88
5.14.2.12	<a href="#">Ak_write_block</a>	88
5.15	<a href="#">file/idx/bitmap.h File Reference</a>	88
5.15.1	<a href="#">Detailed Description</a>	89
5.15.2	<a href="#">Function Documentation</a>	89
5.15.2.1	<a href="#">AK_add_to_bitmap_index</a>	89
5.15.2.2	<a href="#">Ak_bitmap_test</a>	90
5.15.2.3	<a href="#">Ak_create_Index</a>	90
5.15.2.4	<a href="#">AK_create_Index_Table</a>	90
5.15.2.5	<a href="#">AK_delete_bitmap_index</a>	91
5.15.2.6	<a href="#">Ak_get_Attribute</a>	91
5.15.2.7	<a href="#">AK_get_Attribute</a>	91
5.15.2.8	<a href="#">Ak_If_ExistOp</a>	92
5.15.2.9	<a href="#">Ak_print_Att_Test</a>	92
5.15.2.10	<a href="#">Ak_print_Header_Test</a>	92
5.15.2.11	<a href="#">AK_update</a>	93
5.15.2.12	<a href="#">Ak_write_block</a>	93
5.16	<a href="#">file/idx/btree.c File Reference</a>	93
5.16.1	<a href="#">Detailed Description</a>	94
5.16.2	<a href="#">Function Documentation</a>	94
5.16.2.1	<a href="#">AK_btree_create</a>	94
5.16.2.2	<a href="#">AK_btree_search_delete</a>	94
5.17	<a href="#">file/idx/btree.h File Reference</a>	94
5.17.1	<a href="#">Detailed Description</a>	95
5.17.2	<a href="#">Function Documentation</a>	95
5.17.2.1	<a href="#">AK_btree_create</a>	95
5.17.2.2	<a href="#">AK_btree_search_delete</a>	95
5.18	<a href="#">file/idx/hash.c File Reference</a>	96
5.18.1	<a href="#">Detailed Description</a>	96
5.18.2	<a href="#">Function Documentation</a>	96
5.18.2.1	<a href="#">AK_change_hash_info</a>	96
5.18.2.2	<a href="#">AK_create_hash_index</a>	97
5.18.2.3	<a href="#">AK_delete_in_hash_index</a>	97

5.18.2.4	<a href="#">AK_elem_hash_value</a>	97
5.18.2.5	<a href="#">AK_find_delete_in_hash_index</a>	98
5.18.2.6	<a href="#">AK_find_in_hash_index</a>	98
5.18.2.7	<a href="#">AK_get_hash_info</a>	98
5.18.2.8	<a href="#">Ak_get_nth_main_bucket_add</a>	99
5.18.2.9	<a href="#">Ak_hash_test</a>	99
5.18.2.10	<a href="#">Ak_insert_bucket_to_block</a>	99
5.18.2.11	<a href="#">AK_insert_in_hash_index</a>	100
5.18.2.12	<a href="#">Ak_update_bucket_in_block</a>	100
5.19	<a href="#">file/idx/hash.h File Reference</a>	100
5.19.1	<a href="#">Detailed Description</a>	101
5.19.2	<a href="#">Function Documentation</a>	101
5.19.2.1	<a href="#">AK_change_hash_info</a>	101
5.19.2.2	<a href="#">AK_create_hash_index</a>	102
5.19.2.3	<a href="#">AK_delete_in_hash_index</a>	102
5.19.2.4	<a href="#">AK_elem_hash_value</a>	102
5.19.2.5	<a href="#">AK_find_delete_in_hash_index</a>	102
5.19.2.6	<a href="#">AK_find_in_hash_index</a>	103
5.19.2.7	<a href="#">AK_get_hash_info</a>	103
5.19.2.8	<a href="#">Ak_get_nth_main_bucket_add</a>	103
5.19.2.9	<a href="#">Ak_hash_test</a>	104
5.19.2.10	<a href="#">Ak_insert_bucket_to_block</a>	104
5.19.2.11	<a href="#">AK_insert_in_hash_index</a>	104
5.19.2.12	<a href="#">Ak_update_bucket_in_block</a>	105
5.20	<a href="#">file/idx/index.c File Reference</a>	105
5.20.1	<a href="#">Detailed Description</a>	106
5.20.2	<a href="#">Function Documentation</a>	106
5.20.2.1	<a href="#">Ak_Delete_All_elementsAd</a>	106
5.20.2.2	<a href="#">Ak_Delete_elementAd</a>	106
5.20.2.3	<a href="#">Ak_Get_First_elementAd</a>	106
5.20.2.4	<a href="#">AK_get_index_header</a>	107
5.20.2.5	<a href="#">AK_get_index_num_records</a>	107
5.20.2.6	<a href="#">AK_get_index_tuple</a>	108
5.20.2.7	<a href="#">Ak_Get_Last_elementAd</a>	108
5.20.2.8	<a href="#">Ak_Get_Next_elementAd</a>	108
5.20.2.9	<a href="#">Ak_Get_Position_Of_elementAd</a>	108
5.20.2.10	<a href="#">Ak_Get_Previous_elementAd</a>	109
5.20.2.11	<a href="#">AK_index_table_exist</a>	109
5.20.2.12	<a href="#">AK_index_test</a>	109
5.20.2.13	<a href="#">Ak_InitializelistAd</a>	110

5.20.2.14 Ak_Insert_NewelementAd . . . . .	110
5.20.2.15 AK_num_index_attr . . . . .	110
5.20.2.16 AK_print_index_table . . . . .	110
5.21 file/idx/index.h File Reference . . . . .	111
5.21.1 Detailed Description . . . . .	112
5.21.2 Function Documentation . . . . .	112
5.21.2.1 Ak_Delete_All_elementsAd . . . . .	112
5.21.2.2 Ak_Delete_elementAd . . . . .	112
5.21.2.3 Ak_Get_First_elementAd . . . . .	112
5.21.2.4 AK_get_index_num_records . . . . .	113
5.21.2.5 AK_get_index_tuple . . . . .	113
5.21.2.6 Ak_Get_Last_elementAd . . . . .	113
5.21.2.7 Ak_Get_Next_elementAd . . . . .	114
5.21.2.8 Ak_Get_Position_Of_elementAd . . . . .	114
5.21.2.9 Ak_Get_Previous_elementAd . . . . .	114
5.21.2.10 AK_index_table_exist . . . . .	115
5.21.2.11 AK_index_test . . . . .	115
5.21.2.12 Ak_InitializelistAd . . . . .	115
5.21.2.13 Ak_Insert_NewelementAd . . . . .	116
5.21.2.14 AK_num_index_attr . . . . .	116
5.21.2.15 AK_print_index_table . . . . .	116
5.22 file/table.c File Reference . . . . .	116
5.22.1 Detailed Description . . . . .	118
5.22.2 Function Documentation . . . . .	118
5.22.2.1 AK_check_tables_scheme . . . . .	118
5.22.2.2 AK_get_attr_index . . . . .	118
5.22.2.3 AK_get_attr_name . . . . .	118
5.22.2.4 AK_get_column . . . . .	119
5.22.2.5 AK_get_header . . . . .	119
5.22.2.6 AK_get_num_records . . . . .	119
5.22.2.7 AK_get_row . . . . .	120
5.22.2.8 AK_get_table_obj_id . . . . .	120
5.22.2.9 AK_get_tuple . . . . .	121
5.22.2.10 AK_num_attr . . . . .	121
5.22.2.11 AK_op_rename_test . . . . .	121
5.22.2.12 AK_print_row . . . . .	121
5.22.2.13 AK_print_row_spacer . . . . .	122
5.22.2.14 AK_print_row_spacer_to_file . . . . .	122
5.22.2.15 AK_print_row_to_file . . . . .	122
5.22.2.16 AK_print_table . . . . .	123

5.22.2.17 AK_print_table_to_file . . . . .	123
5.22.2.18 AK_rename . . . . .	123
5.22.2.19 AK_table_empty . . . . .	124
5.22.2.20 AK_table_exist . . . . .	124
5.22.2.21 AK_table_test . . . . .	124
5.22.2.22 AK_tuple_to_string . . . . .	125
5.22.2.23 get_row_attr_data . . . . .	125
5.23 file/table.h File Reference . . . . .	125
5.23.1 Detailed Description . . . . .	127
5.23.2 Function Documentation . . . . .	127
5.23.2.1 AK_check_tables_scheme . . . . .	127
5.23.2.2 AK_get_attr_index . . . . .	127
5.23.2.3 AK_get_attr_name . . . . .	127
5.23.2.4 AK_get_column . . . . .	128
5.23.2.5 AK_get_header . . . . .	128
5.23.2.6 AK_get_num_records . . . . .	128
5.23.2.7 AK_get_row . . . . .	129
5.23.2.8 AK_get_table_obj_id . . . . .	129
5.23.2.9 AK_get_tuple . . . . .	130
5.23.2.10 AK_num_attr . . . . .	130
5.23.2.11 AK_op_rename_test . . . . .	130
5.23.2.12 AK_print_row . . . . .	130
5.23.2.13 AK_print_row_spacer . . . . .	131
5.23.2.14 AK_print_row_spacer_to_file . . . . .	131
5.23.2.15 AK_print_row_to_file . . . . .	131
5.23.2.16 AK_print_table . . . . .	132
5.23.2.17 AK_print_table_to_file . . . . .	132
5.23.2.18 AK_rename . . . . .	132
5.23.2.19 AK_table_empty . . . . .	133
5.23.2.20 AK_table_test . . . . .	133
5.23.2.21 AK_tuple_to_string . . . . .	133
5.23.2.22 get_row_attr_data . . . . .	134
5.24 file/test.c File Reference . . . . .	134
5.24.1 Detailed Description . . . . .	134
5.24.2 Function Documentation . . . . .	135
5.24.2.1 AK_create_test_tables . . . . .	135
5.24.2.2 AK_get_table_attribute_types . . . . .	135
5.24.2.3 create_header_test . . . . .	135
5.24.2.4 get_column_test . . . . .	135
5.24.2.5 get_row_test . . . . .	136

5.24.2.6	<a href="#">insert_data_test</a>	136
5.24.2.7	<a href="#">selection_test</a>	136
5.25	<a href="#">file/test.h File Reference</a>	137
5.25.1	<a href="#">Detailed Description</a>	137
5.25.2	<a href="#">Function Documentation</a>	137
5.25.2.1	<a href="#">AK_create_test_tables</a>	137
5.25.2.2	<a href="#">AK_get_table_attribute_types</a>	138
5.25.2.3	<a href="#">create_header_test</a>	138
5.25.2.4	<a href="#">get_column_test</a>	138
5.25.2.5	<a href="#">get_row_test</a>	139
5.25.2.6	<a href="#">insert_data_test</a>	139
5.25.2.7	<a href="#">selection_test</a>	139
5.26	<a href="#">mm/memoman.c File Reference</a>	140
5.26.1	<a href="#">Detailed Description</a>	141
5.26.2	<a href="#">Function Documentation</a>	141
5.26.2.1	<a href="#">AK_cache_AK_malloc</a>	141
5.26.2.2	<a href="#">AK_cache_block</a>	141
5.26.2.3	<a href="#">AK_cache_result</a>	141
5.26.2.4	<a href="#">AK_find_AK_free_space</a>	142
5.26.2.5	<a href="#">AK_find_available_result_block</a>	142
5.26.2.6	<a href="#">AK_flush_cache</a>	142
5.26.2.7	<a href="#">AK_generate_result_id</a>	142
5.26.2.8	<a href="#">AK_get_block</a>	143
5.26.2.9	<a href="#">AK_get_index_addresses</a>	143
5.26.2.10	<a href="#">AK_get_index_segment_addresses</a>	143
5.26.2.11	<a href="#">AK_get_segment_addresses</a>	144
5.26.2.12	<a href="#">AK_get_table_addresses</a>	144
5.26.2.13	<a href="#">AK_init_new_extent</a>	144
5.26.2.14	<a href="#">AK_mem_block_modify</a>	145
5.26.2.15	<a href="#">AK_memoman_init</a>	145
5.26.2.16	<a href="#">AK_query_mem_AK_malloc</a>	145
5.26.2.17	<a href="#">AK_redo_log_AK_malloc</a>	145
5.26.2.18	<a href="#">AK_refresh_cache</a>	146
5.27	<a href="#">mm/memoman.h File Reference</a>	146
5.27.1	<a href="#">Detailed Description</a>	147
5.27.2	<a href="#">Function Documentation</a>	147
5.27.2.1	<a href="#">AK_cache_AK_malloc</a>	147
5.27.2.2	<a href="#">AK_cache_block</a>	148
5.27.2.3	<a href="#">AK_cache_result</a>	148
5.27.2.4	<a href="#">AK_find_AK_free_space</a>	148

5.27.2.5	<a href="#">AK_find_available_result_block</a>	149
5.27.2.6	<a href="#">AK_flush_cache</a>	149
5.27.2.7	<a href="#">AK_generate_result_id</a>	149
5.27.2.8	<a href="#">AK_get_block</a>	149
5.27.2.9	<a href="#">AK_get_index_addresses</a>	150
5.27.2.10	<a href="#">AK_get_index_segment_addresses</a>	150
5.27.2.11	<a href="#">AK_get_segment_addresses</a>	150
5.27.2.12	<a href="#">AK_get_table_addresses</a>	150
5.27.2.13	<a href="#">AK_init_new_extent</a>	151
5.27.2.14	<a href="#">AK_mem_block_modify</a>	151
5.27.2.15	<a href="#">AK_memoman_init</a>	151
5.27.2.16	<a href="#">AK_query_mem_AK_malloc</a>	152
5.27.2.17	<a href="#">AK_redo_log_AK_malloc</a>	152
5.27.2.18	<a href="#">AK_refresh_cache</a>	152
5.28	<a href="#">opti/query_optimization.c File Reference</a>	152
5.28.1	<a href="#">Detailed Description</a>	153
5.28.2	<a href="#">Function Documentation</a>	153
5.28.2.1	<a href="#">AK_execute_rel_eq</a>	153
5.28.2.2	<a href="#">AK_print_optimized_query</a>	153
5.28.2.3	<a href="#">AK_query_optimization</a>	153
5.28.2.4	<a href="#">AK_query_optimization_test</a>	154
5.29	<a href="#">opti/query_optimization.h File Reference</a>	154
5.29.1	<a href="#">Detailed Description</a>	155
5.29.2	<a href="#">Function Documentation</a>	155
5.29.2.1	<a href="#">AK_execute_rel_eq</a>	155
5.29.2.2	<a href="#">AK_print_optimized_query</a>	155
5.29.2.3	<a href="#">AK_query_optimization</a>	155
5.29.2.4	<a href="#">AK_query_optimization_test</a>	156
5.30	<a href="#">opti/rel_eq_assoc.c File Reference</a>	156
5.30.1	<a href="#">Detailed Description</a>	156
5.30.2	<a href="#">Function Documentation</a>	157
5.30.2.1	<a href="#">AK_compare</a>	157
5.30.2.2	<a href="#">AK_print_rel_eq_assoc</a>	157
5.30.2.3	<a href="#">AK_rel_eq_assoc</a>	157
5.30.2.4	<a href="#">AK_rel_eq_assoc_test</a>	157
5.31	<a href="#">opti/rel_eq_assoc.h File Reference</a>	158
5.31.1	<a href="#">Detailed Description</a>	158
5.31.2	<a href="#">Function Documentation</a>	158
5.31.2.1	<a href="#">AK_compare</a>	158
5.31.2.2	<a href="#">AK_print_rel_eq_assoc</a>	159

5.31.2.3	<a href="#">AK_rel_eq_assoc</a>	159
5.31.2.4	<a href="#">AK_rel_eq_assoc_test</a>	159
5.32	<a href="#">opti/rel_eq_comut.c File Reference</a>	159
5.32.1	Detailed Description	160
5.32.2	Function Documentation	160
5.32.2.1	<a href="#">AK_print_rel_eq_comut</a>	160
5.32.2.2	<a href="#">AK_rel_eq_commute_with_theta_join</a>	160
5.32.2.3	<a href="#">AK_rel_eq_comut</a>	161
5.32.2.4	<a href="#">AK_rel_eq_comut_test</a>	161
5.33	<a href="#">opti/rel_eq_comut.h File Reference</a>	161
5.33.1	Detailed Description	161
5.33.2	Function Documentation	162
5.33.2.1	<a href="#">AK_print_rel_eq_comut</a>	162
5.33.2.2	<a href="#">AK_rel_eq_commute_with_theta_join</a>	162
5.33.2.3	<a href="#">AK_rel_eq_comut</a>	162
5.33.2.4	<a href="#">AK_rel_eq_comut_test</a>	163
5.34	<a href="#">opti/rel_eq_projection.c File Reference</a>	163
5.34.1	Detailed Description	163
5.34.2	Function Documentation	163
5.34.2.1	<a href="#">AK_print_rel_eq_projection</a>	163
5.34.2.2	<a href="#">AK_rel_eq_can_commute</a>	164
5.34.2.3	<a href="#">AK_rel_eq_collect_cond_attributes</a>	164
5.34.2.4	<a href="#">AK_rel_eq_get_attributes</a>	164
5.34.2.5	<a href="#">AK_rel_eq_is_subset</a>	165
5.34.2.6	<a href="#">AK_rel_eq_projection</a>	166
5.34.2.7	<a href="#">AK_rel_eq_projection_attributes</a>	166
5.34.2.8	<a href="#">AK_rel_eq_projection_test</a>	166
5.34.2.9	<a href="#">AK_rel_eq_remove_duplicates</a>	166
5.35	<a href="#">opti/rel_eq_projection.h File Reference</a>	167
5.35.1	Detailed Description	167
5.35.2	Function Documentation	167
5.35.2.1	<a href="#">AK_print_rel_eq_projection</a>	167
5.35.2.2	<a href="#">AK_rel_eq_can_commute</a>	168
5.35.2.3	<a href="#">AK_rel_eq_collect_cond_attributes</a>	168
5.35.2.4	<a href="#">AK_rel_eq_get_attributes</a>	168
5.35.2.5	<a href="#">AK_rel_eq_is_subset</a>	169
5.35.2.6	<a href="#">AK_rel_eq_projection</a>	170
5.35.2.7	<a href="#">AK_rel_eq_projection_attributes</a>	170
5.35.2.8	<a href="#">AK_rel_eq_projection_test</a>	170
5.35.2.9	<a href="#">AK_rel_eq_remove_duplicates</a>	170



5.36	<a href="#">opti/rel_eq_selection.c File Reference</a>	171
5.36.1	<a href="#">Detailed Description</a>	171
5.36.2	<a href="#">Function Documentation</a>	171
5.36.2.1	<a href="#">AK_print_rel_eq_selection</a>	171
5.36.2.2	<a href="#">AK_rel_eq_cond_attributes</a>	172
5.36.2.3	<a href="#">AK_rel_eq_get_attributes_char</a>	172
5.36.2.4	<a href="#">AK_rel_eq_is_attr_subset</a>	172
5.36.2.5	<a href="#">AK_rel_eq_selection</a>	173
5.36.2.6	<a href="#">AK_rel_eq_selection_test</a>	173
5.36.2.7	<a href="#">AK_rel_eq_share_attributes</a>	174
5.36.2.8	<a href="#">AK_rel_eq_split_condition</a>	174
5.37	<a href="#">opti/rel_eq_selection.h File Reference</a>	175
5.37.1	<a href="#">Detailed Description</a>	175
5.37.2	<a href="#">Function Documentation</a>	175
5.37.2.1	<a href="#">AK_print_rel_eq_selection</a>	175
5.37.2.2	<a href="#">AK_rel_eq_cond_attributes</a>	176
5.37.2.3	<a href="#">AK_rel_eq_get_attributes_char</a>	176
5.37.2.4	<a href="#">AK_rel_eq_is_attr_subset</a>	176
5.37.2.5	<a href="#">AK_rel_eq_selection</a>	177
5.37.2.6	<a href="#">AK_rel_eq_selection_test</a>	177
5.37.2.7	<a href="#">AK_rel_eq_share_attributes</a>	178
5.37.2.8	<a href="#">AK_rel_eq_split_condition</a>	178
5.38	<a href="#">rec/archive_log.h File Reference</a>	179
5.38.1	<a href="#">Detailed Description</a>	179
5.38.2	<a href="#">Function Documentation</a>	179
5.38.2.1	<a href="#">AK_archive_log</a>	179
5.38.2.2	<a href="#">AK_empty_archive_log</a>	179
5.38.2.3	<a href="#">AK_get_timestamp</a>	180
5.39	<a href="#">rec/recovery.c File Reference</a>	180
5.39.1	<a href="#">Detailed Description</a>	180
5.39.2	<a href="#">Function Documentation</a>	180
5.39.2.1	<a href="#">AK_recover_archive_log</a>	180
5.39.2.2	<a href="#">AK_recover_operation</a>	181
5.39.2.3	<a href="#">AK_recovery_insert_row</a>	181
5.39.2.4	<a href="#">AK_recovery_test</a>	181
5.39.2.5	<a href="#">AK_recovery_tokenize</a>	182
5.39.3	<a href="#">Variable Documentation</a>	182
5.39.3.1	<a href="#">grandfailure</a>	182
5.40	<a href="#">rec/redo_log.c File Reference</a>	182
5.40.1	<a href="#">Detailed Description</a>	182

5.40.2	Function Documentation	182
5.40.2.1	AK_add_to_redolog	182
5.40.2.2	AK_check_attributes	183
5.40.2.3	AK_printout_redolog	183
5.41	rel/aggregation.c File Reference	183
5.41.1	Detailed Description	184
5.41.2	Function Documentation	184
5.41.2.1	AK_agg_input_add	184
5.41.2.2	AK_agg_input_add_to_beginning	184
5.41.2.3	AK_agg_input_fix	185
5.41.2.4	AK_agg_input_init	185
5.41.2.5	AK_aggregation	185
5.41.2.6	Ak_aggregation_test	186
5.41.2.7	AK_header_size	186
5.41.2.8	AK_search_unsorted	186
5.42	rel/aggregation.h File Reference	187
5.42.1	Detailed Description	188
5.42.2	Function Documentation	188
5.42.2.1	AK_agg_input_add	188
5.42.2.2	AK_agg_input_add_to_beginning	188
5.42.2.3	AK_agg_input_fix	189
5.42.2.4	AK_agg_input_init	189
5.42.2.5	AK_aggregation	189
5.42.2.6	Ak_aggregation_test	190
5.42.2.7	AK_header_size	190
5.43	rel/difference.c File Reference	190
5.43.1	Detailed Description	191
5.43.2	Function Documentation	191
5.43.2.1	AK_difference	191
5.43.2.2	Ak_op_difference_test	191
5.44	rel/difference.h File Reference	191
5.44.1	Detailed Description	192
5.44.2	Function Documentation	192
5.44.2.1	AK_difference	192
5.44.2.2	Ak_op_difference_test	192
5.45	rel/expression_check.c File Reference	192
5.45.1	Detailed Description	193
5.45.2	Function Documentation	193
5.45.2.1	AK_check_arithmetic_statement	193
5.45.2.2	AK_check_if_row_satisfies_expression	193

5.45.2.3	<a href="#">Ak_check_regex_expression</a>	194
5.45.2.4	<a href="#">Ak_check_regex_operator_expression</a>	194
5.45.2.5	<a href="#">AK_replace_wild_card</a>	194
5.46	<a href="#">rel/expression_check.h File Reference</a>	195
5.46.1	<a href="#">Detailed Description</a>	195
5.46.2	<a href="#">Function Documentation</a>	195
5.46.2.1	<a href="#">AK_check_arithmetic_statement</a>	195
5.46.2.2	<a href="#">AK_check_if_row_satisfies_expression</a>	196
5.46.2.3	<a href="#">Ak_check_regex_expression</a>	196
5.46.2.4	<a href="#">Ak_check_regex_operator_expression</a>	196
5.47	<a href="#">rel/intersect.c File Reference</a>	196
5.47.1	<a href="#">Detailed Description</a>	197
5.47.2	<a href="#">Function Documentation</a>	197
5.47.2.1	<a href="#">AK_intersect</a>	197
5.47.2.2	<a href="#">Ak_op_intersect_test</a>	197
5.48	<a href="#">rel/intersect.h File Reference</a>	197
5.48.1	<a href="#">Detailed Description</a>	198
5.48.2	<a href="#">Function Documentation</a>	198
5.48.2.1	<a href="#">AK_intersect</a>	198
5.48.2.2	<a href="#">Ak_op_intersect_test</a>	198
5.49	<a href="#">rel/nat_join.c File Reference</a>	198
5.49.1	<a href="#">Detailed Description</a>	199
5.49.2	<a href="#">Function Documentation</a>	199
5.49.2.1	<a href="#">AK_copy_blocks_join</a>	199
5.49.2.2	<a href="#">AK_create_join_block_header</a>	199
5.49.2.3	<a href="#">AK_join</a>	200
5.49.2.4	<a href="#">AK_merge_block_join</a>	200
5.49.2.5	<a href="#">AK_op_join_test</a>	200
5.50	<a href="#">rel/nat_join.h File Reference</a>	201
5.50.1	<a href="#">Detailed Description</a>	201
5.50.2	<a href="#">Function Documentation</a>	201
5.50.2.1	<a href="#">AK_copy_blocks_join</a>	201
5.50.2.2	<a href="#">AK_create_join_block_header</a>	202
5.50.2.3	<a href="#">AK_join</a>	202
5.50.2.4	<a href="#">AK_merge_block_join</a>	202
5.50.2.5	<a href="#">AK_op_join_test</a>	203
5.51	<a href="#">rel/product.c File Reference</a>	203
5.51.1	<a href="#">Detailed Description</a>	203
5.51.2	<a href="#">Function Documentation</a>	203
5.51.2.1	<a href="#">AK_op_product_test</a>	203

5.51.2.2	AK_product	204
5.52	rel/product.h File Reference	204
5.52.1	Detailed Description	204
5.52.2	Function Documentation	204
5.52.2.1	AK_op_product_test	204
5.52.2.2	AK_product	205
5.53	rel/projection.c File Reference	205
5.53.1	Detailed Description	206
5.53.2	Function Documentation	206
5.53.2.1	AK_copy_block_projection	206
5.53.2.2	AK_create_block_header	206
5.53.2.3	AK_create_header_name	207
5.53.2.4	AK_determine_header_type	208
5.53.2.5	AK_get_operator	208
5.53.2.6	AK_op_projection_test	208
5.53.2.7	AK_perform_operatrion	208
5.53.2.8	AK_projection	209
5.53.2.9	AK_temp_create_table	209
5.53.2.10	removeSubstring	209
5.54	rel/projection.h File Reference	210
5.54.1	Detailed Description	210
5.54.2	Function Documentation	211
5.54.2.1	AK_copy_block_projection	211
5.54.2.2	AK_create_block_header	211
5.54.2.3	AK_create_header_name	211
5.54.2.4	AK_determine_header_type	211
5.54.2.5	AK_get_operator	212
5.54.2.6	AK_op_projection_test	212
5.54.2.7	AK_perform_operatrion	212
5.54.2.8	AK_projection	212
5.54.2.9	AK_temp_create_table	213
5.54.2.10	removeSubstring	213
5.55	rel/selection.c File Reference	213
5.55.1	Detailed Description	214
5.55.2	Function Documentation	214
5.55.2.1	AK_op_selection_test	214
5.55.2.2	AK_op_selection_test2	214
5.55.2.3	AK_op_selection_test_redolog	214
5.55.2.4	AK_selection	214
5.56	rel/selection.h File Reference	215

5.56.1 Detailed Description . . . . .	215
5.56.2 Function Documentation . . . . .	215
5.56.2.1 AK_op_selection_test . . . . .	215
5.56.2.2 AK_op_selection_test2 . . . . .	215
5.56.2.3 AK_op_selection_test_redolog . . . . .	215
5.56.2.4 AK_selection . . . . .	216
5.57 rel/sequence.c File Reference . . . . .	216
5.57.1 Detailed Description . . . . .	216
5.57.2 Function Documentation . . . . .	217
5.57.2.1 AK_sequence_add . . . . .	217
5.57.2.2 AK_sequence_current_value . . . . .	217
5.57.2.3 AK_sequence_get_id . . . . .	217
5.57.2.4 AK_sequence_modify . . . . .	218
5.57.2.5 AK_sequence_next_value . . . . .	218
5.57.2.6 AK_sequence_remove . . . . .	218
5.57.2.7 AK_sequence_rename . . . . .	218
5.57.2.8 AK_sequence_test . . . . .	219
5.58 rel/sequence.h File Reference . . . . .	219
5.58.1 Detailed Description . . . . .	219
5.58.2 Function Documentation . . . . .	220
5.58.2.1 AK_sequence_add . . . . .	220
5.58.2.2 AK_sequence_current_value . . . . .	220
5.58.2.3 AK_sequence_get_id . . . . .	220
5.58.2.4 AK_sequence_modify . . . . .	221
5.58.2.5 AK_sequence_next_value . . . . .	221
5.58.2.6 AK_sequence_remove . . . . .	221
5.58.2.7 AK_sequence_rename . . . . .	221
5.58.2.8 AK_sequence_test . . . . .	222
5.59 rel/theta_join.c File Reference . . . . .	222
5.59.1 Detailed Description . . . . .	222
5.59.2 Function Documentation . . . . .	222
5.59.2.1 AK_check_constraints . . . . .	222
5.59.2.2 AK_create_theta_join_header . . . . .	223
5.59.2.3 AK_op_theta_join_test . . . . .	223
5.59.2.4 AK_theta_join . . . . .	223
5.60 rel/theta_join.h File Reference . . . . .	224
5.60.1 Detailed Description . . . . .	224
5.60.2 Function Documentation . . . . .	224
5.60.2.1 AK_check_constraints . . . . .	224
5.60.2.2 AK_create_theta_join_header . . . . .	225

5.60.2.3	AK_op_theta_join_test	225
5.60.2.4	AK_theta_join	225
5.61	rel/union.c File Reference	226
5.61.1	Detailed Description	226
5.61.2	Function Documentation	226
5.61.2.1	AK_op_union_test	226
5.61.2.2	AK_union	226
5.62	rel/union.h File Reference	227
5.62.1	Detailed Description	227
5.62.2	Function Documentation	227
5.62.2.1	AK_op_union_test	227
5.62.2.2	AK_union	227
5.63	sql/cs/between.c File Reference	228
5.63.1	Detailed Description	228
5.63.2	Function Documentation	228
5.63.2.1	AK_constraint_between_test	228
5.63.2.2	AK_delete_constraint_between	229
5.63.2.3	AK_find_table_address	229
5.63.2.4	AK_print_constraints	229
5.63.2.5	AK_read_constraint_between	229
5.63.2.6	AK_set_constraint_between	230
5.64	sql/cs/between.h File Reference	230
5.64.1	Detailed Description	231
5.64.2	Function Documentation	231
5.64.2.1	AK_constraint_between_test	231
5.64.2.2	AK_find_table_address	231
5.64.2.3	AK_read_constraint_between	231
5.64.2.4	AK_set_constraint_between	232
5.65	sql/cs/check_constraint.c File Reference	232
5.65.1	Detailed Description	232
5.65.2	Function Documentation	232
5.65.2.1	AK_check_constraint	232
5.65.2.2	AK_check_constraint_test	233
5.65.2.3	AK_set_check_constraint	233
5.65.2.4	condition_passed	233
5.66	sql/cs/check_constraint.h File Reference	234
5.66.1	Detailed Description	234
5.66.2	Function Documentation	234
5.66.2.1	AK_check_constraint	234
5.66.2.2	AK_check_constraint_test	235

5.66.2.3	<a href="#">AK_set_check_constraint</a>	235
5.66.2.4	<a href="#">condition_passed</a>	235
5.67	<a href="#">sql/cs/constraint_names.c File Reference</a>	236
5.67.1	<a href="#">Detailed Description</a>	236
5.67.2	<a href="#">Function Documentation</a>	236
5.67.2.1	<a href="#">AK_check_constraint_name</a>	236
5.67.2.2	<a href="#">AK_constraint_names_test</a>	236
5.68	<a href="#">sql/cs/constraint_names.h File Reference</a>	237
5.68.1	<a href="#">Detailed Description</a>	237
5.68.2	<a href="#">Function Documentation</a>	237
5.68.2.1	<a href="#">AK_check_constraint_name</a>	237
5.68.2.2	<a href="#">AK_constraint_names_test</a>	237
5.69	<a href="#">sql/cs/nnull.c File Reference</a>	238
5.69.1	<a href="#">Detailed Description</a>	238
5.69.2	<a href="#">Function Documentation</a>	238
5.69.2.1	<a href="#">AK_delete_constraint_not_null</a>	238
5.69.2.2	<a href="#">AK_null_test</a>	238
5.69.2.3	<a href="#">AK_read_constraint_not_null</a>	239
5.69.2.4	<a href="#">AK_set_constraint_not_null</a>	239
5.70	<a href="#">sql/cs/nnull.h File Reference</a>	239
5.70.1	<a href="#">Detailed Description</a>	240
5.70.2	<a href="#">Function Documentation</a>	240
5.70.2.1	<a href="#">AK_delete_constraint_not_null</a>	240
5.70.2.2	<a href="#">AK_null_test</a>	240
5.70.2.3	<a href="#">AK_read_constraint_not_null</a>	240
5.70.2.4	<a href="#">AK_set_constraint_not_null</a>	241
5.71	<a href="#">sql/cs/reference.c File Reference</a>	241
5.71.1	<a href="#">Detailed Description</a>	241
5.71.2	<a href="#">Function Documentation</a>	242
5.71.2.1	<a href="#">AK_add_reference</a>	242
5.71.2.2	<a href="#">AK_get_reference</a>	242
5.71.2.3	<a href="#">AK_reference_check_attribute</a>	242
5.71.2.4	<a href="#">AK_reference_check_entry</a>	243
5.71.2.5	<a href="#">AK_reference_check_if_update_needed</a>	243
5.71.2.6	<a href="#">AK_reference_check_restricion</a>	243
5.71.2.7	<a href="#">AK_reference_test</a>	244
5.71.2.8	<a href="#">AK_reference_update</a>	244
5.72	<a href="#">sql/cs/reference.h File Reference</a>	244
5.72.1	<a href="#">Detailed Description</a>	246
5.72.2	<a href="#">Macro Definition Documentation</a>	246

5.72.2.1	REF_TYPE_NO_ACTION	246
5.72.3	Function Documentation	246
5.72.3.1	AK_add_reference	246
5.72.3.2	Ak_delete_row	246
5.72.3.3	AK_get_reference	247
5.72.3.4	AK_initialize_new_segment	247
5.72.3.5	Ak_Insert_New_Element	247
5.72.3.6	Ak_Insert_New_Element_For_Update	248
5.72.3.7	Ak_insert_row	248
5.72.3.8	AK_reference_check_attribute	249
5.72.3.9	AK_reference_check_entry	249
5.72.3.10	AK_reference_check_if_update_needed	249
5.72.3.11	AK_reference_check_restricion	249
5.72.3.12	AK_reference_test	250
5.72.3.13	AK_reference_update	250
5.72.3.14	AK_selection	250
5.72.3.15	Ak_update_row	251
5.73	sql/cs/unique.c File Reference	251
5.73.1	Detailed Description	251
5.73.2	Function Documentation	251
5.73.2.1	AK_delete_constraint_unique	251
5.73.2.2	AK_read_constraint_unique	252
5.73.2.3	Ak_set_constraint_unique	252
5.73.2.4	AK_unique_test	252
5.74	sql/cs/unique.h File Reference	253
5.74.1	Detailed Description	253
5.74.2	Function Documentation	253
5.74.2.1	AK_delete_constraint_unique	253
5.74.2.2	AK_read_constraint_unique	253
5.74.2.3	Ak_set_constraint_unique	254
5.74.2.4	AK_unique_test	254
5.75	sql/drop.c File Reference	254
5.75.1	Detailed Description	255
5.75.2	Function Documentation	255
5.75.2.1	AK_drop	255
5.75.2.2	AK_drop_help_function	255
5.75.2.3	AK_drop_test	256
5.75.2.4	AK_if_exist	256
5.75.3	Variable Documentation	256
5.75.3.1	system_catalog	256



5.76	sql/drop.h File Reference	257
5.76.1	Function Documentation	257
5.76.1.1	AK_drop	257
5.76.1.2	AK_drop_test	257
5.76.1.3	AK_if_exist	258
5.77	sql/function.c File Reference	258
5.77.1	Detailed Description	258
5.77.2	Function Documentation	259
5.77.2.1	AK_check_function_arguments	259
5.77.2.2	AK_check_function_arguments_type	259
5.77.2.3	AK_function_add	259
5.77.2.4	AK_function_arguments_add	260
5.77.2.5	AK_function_arguments_remove_by_obj_id	260
5.77.2.6	AK_function_change_return_type	260
5.77.2.7	AK_function_remove_by_name	261
5.77.2.8	AK_function_remove_by_obj_id	261
5.77.2.9	AK_function_rename	261
5.77.2.10	AK_function_test	261
5.77.2.11	AK_get_function_obj_id	262
5.78	sql/function.h File Reference	262
5.78.1	Detailed Description	263
5.78.2	Function Documentation	263
5.78.2.1	AK_check_function_arguments	263
5.78.2.2	AK_check_function_arguments_type	263
5.78.2.3	AK_function_add	263
5.78.2.4	AK_function_arguments_add	264
5.78.2.5	AK_function_arguments_remove_by_obj_id	264
5.78.2.6	AK_function_change_return_type	264
5.78.2.7	AK_function_remove_by_name	265
5.78.2.8	AK_function_remove_by_obj_id	265
5.78.2.9	AK_function_rename	265
5.78.2.10	AK_function_test	266
5.78.2.11	AK_get_function_obj_id	266
5.79	sql/privileges.c File Reference	266
5.79.1	Detailed Description	267
5.79.2	Function Documentation	267
5.79.2.1	AK_add_user_to_group	267
5.79.2.2	AK_check_group_privilege	268
5.79.2.3	AK_check_privilege	268
5.79.2.4	AK_check_user_privilege	268

5.79.2.5	AK_grant_privilege_group . . . . .	269
5.79.2.6	AK_grant_privilege_user . . . . .	269
5.79.2.7	AK_group_add . . . . .	269
5.79.2.8	AK_group_get_id . . . . .	270
5.79.2.9	AK_group_remove_by_name . . . . .	270
5.79.2.10	AK_group_rename . . . . .	270
5.79.2.11	AK_privileges_test . . . . .	271
5.79.2.12	AK_remove_all_users_from_group . . . . .	271
5.79.2.13	AK_remove_user_from_all_groups . . . . .	271
5.79.2.14	AK_revoke_all_privileges_group . . . . .	271
5.79.2.15	AK_revoke_all_privileges_user . . . . .	272
5.79.2.16	AK_revoke_privilege_group . . . . .	272
5.79.2.17	AK_revoke_privilege_user . . . . .	272
5.79.2.18	AK_user_add . . . . .	273
5.79.2.19	AK_user_get_id . . . . .	273
5.79.2.20	AK_user_remove_by_name . . . . .	273
5.79.2.21	AK_user_rename . . . . .	274
5.80	sql/select.c File Reference . . . . .	274
5.80.1	Detailed Description . . . . .	274
5.80.2	Function Documentation . . . . .	274
5.80.2.1	AK_select . . . . .	274
5.80.2.2	AK_select_test . . . . .	275
5.81	sql/trigger.c File Reference . . . . .	275
5.81.1	Detailed Description . . . . .	276
5.81.2	Function Documentation . . . . .	276
5.81.2.1	AK_trigger_add . . . . .	276
5.81.2.2	AK_trigger_edit . . . . .	276
5.81.2.3	AK_trigger_get_conditions . . . . .	277
5.81.2.4	AK_trigger_get_id . . . . .	277
5.81.2.5	AK_trigger_remove_by_name . . . . .	277
5.81.2.6	AK_trigger_remove_by_obj_id . . . . .	277
5.81.2.7	AK_trigger_rename . . . . .	278
5.81.2.8	AK_trigger_save_conditions . . . . .	278
5.81.2.9	AK_trigger_test . . . . .	278
5.82	sql/trigger.h File Reference . . . . .	279
5.82.1	Detailed Description . . . . .	279
5.82.2	Function Documentation . . . . .	279
5.82.2.1	AK_trigger_add . . . . .	279
5.82.2.2	AK_trigger_edit . . . . .	280
5.82.2.3	AK_trigger_get_conditions . . . . .	280

5.82.2.4	<a href="#">AK_trigger_get_id</a>	280
5.82.2.5	<a href="#">AK_trigger_remove_by_name</a>	281
5.82.2.6	<a href="#">AK_trigger_remove_by_obj_id</a>	281
5.82.2.7	<a href="#">AK_trigger_rename</a>	281
5.82.2.8	<a href="#">AK_trigger_save_conditions</a>	282
5.82.2.9	<a href="#">AK_trigger_test</a>	282
5.83	<a href="#">sql/view.c File Reference</a>	282
5.83.1	<a href="#">Detailed Description</a>	283
5.83.2	<a href="#">Function Documentation</a>	283
5.83.2.1	<a href="#">AK_get_rel_exp</a>	283
5.83.2.2	<a href="#">AK_get_view_obj_id</a>	283
5.83.2.3	<a href="#">AK_get_view_query</a>	283
5.83.2.4	<a href="#">AK_view_add</a>	283
5.83.2.5	<a href="#">AK_view_change_query</a>	284
5.83.2.6	<a href="#">AK_view_remove_by_name</a>	284
5.83.2.7	<a href="#">AK_view_remove_by_obj_id</a>	284
5.83.2.8	<a href="#">AK_view_rename</a>	285
5.83.2.9	<a href="#">AK_view_test</a>	285
5.84	<a href="#">trans/transaction.c File Reference</a>	285
5.84.1	<a href="#">Detailed Description</a>	287
5.84.2	<a href="#">Function Documentation</a>	287
5.84.2.1	<a href="#">AK_acquire_lock</a>	287
5.84.2.2	<a href="#">AK_add_hash_entry_list</a>	287
5.84.2.3	<a href="#">AK_add_lock</a>	288
5.84.2.4	<a href="#">AK_all_transactions_finished</a>	288
5.84.2.5	<a href="#">AK_create_lock</a>	288
5.84.2.6	<a href="#">AK_create_new_transaction_thread</a>	289
5.84.2.7	<a href="#">AK_delete_hash_entry_list</a>	289
5.84.2.8	<a href="#">AK_delete_lock_entry_list</a>	289
5.84.2.9	<a href="#">AK_execute_commands</a>	289
5.84.2.10	<a href="#">AK_execute_transaction</a>	290
5.84.2.11	<a href="#">AK_get_memory_blocks</a>	290
5.84.2.12	<a href="#">AK_handle_observable_transaction_action</a>	290
5.84.2.13	<a href="#">AK_init_observable_transaction</a>	291
5.84.2.14	<a href="#">AK_init_observer_lock</a>	291
5.84.2.15	<a href="#">AK_isLock_waiting</a>	291
5.84.2.16	<a href="#">AK_lock_released</a>	291
5.84.2.17	<a href="#">AK_memory_block_hash</a>	292
5.84.2.18	<a href="#">AK_on_all_transactions_end</a>	292
5.84.2.19	<a href="#">AK_on_lock_release</a>	292

5.84.2.20	<a href="#">AK_on_observable_notify</a>	292
5.84.2.21	<a href="#">AK_on_transaction_end</a>	292
5.84.2.22	<a href="#">AK_release_locks</a>	293
5.84.2.23	<a href="#">AK_remove_transaction_thread</a>	293
5.84.2.24	<a href="#">AK_search_empty_link_for_hook</a>	293
5.84.2.25	<a href="#">AK_search_existing_link_for_hook</a>	294
5.84.2.26	<a href="#">AK_search_lock_entry_list_by_key</a>	294
5.84.2.27	<a href="#">AK_transaction_finished</a>	294
5.84.2.28	<a href="#">AK_transaction_manager</a>	294
5.84.2.29	<a href="#">AK_transaction_register_observer</a>	295
5.84.2.30	<a href="#">AK_transaction_unregister_observer</a>	295
5.84.2.31	<a href="#">handle_transaction_notify</a>	295
5.85	<a href="#">trans/transaction.h File Reference</a>	296
5.85.1	<a href="#">Detailed Description</a>	298
5.85.2	<a href="#">Enumeration Type Documentation</a>	298
5.85.2.1	<a href="#">NoticeType</a>	298
5.85.3	<a href="#">Function Documentation</a>	298
5.85.3.1	<a href="#">AK_acquire_lock</a>	298
5.85.3.2	<a href="#">AK_add_hash_entry_list</a>	300
5.85.3.3	<a href="#">AK_add_lock</a>	300
5.85.3.4	<a href="#">AK_all_transactions_finished</a>	300
5.85.3.5	<a href="#">AK_create_lock</a>	301
5.85.3.6	<a href="#">AK_create_new_transaction_thread</a>	301
5.85.3.7	<a href="#">AK_delete_hash_entry_list</a>	301
5.85.3.8	<a href="#">AK_delete_lock_entry_list</a>	301
5.85.3.9	<a href="#">AK_execute_commands</a>	302
5.85.3.10	<a href="#">AK_execute_transaction</a>	302
5.85.3.11	<a href="#">AK_get_memory_blocks</a>	302
5.85.3.12	<a href="#">AK_handle_observable_transaction_action</a>	303
5.85.3.13	<a href="#">AK_init_observable_transaction</a>	303
5.85.3.14	<a href="#">AK_init_observer_lock</a>	303
5.85.3.15	<a href="#">AK_isLock_waiting</a>	303
5.85.3.16	<a href="#">AK_lock_released</a>	304
5.85.3.17	<a href="#">AK_memory_block_hash</a>	304
5.85.3.18	<a href="#">AK_on_all_transactions_end</a>	304
5.85.3.19	<a href="#">AK_on_lock_release</a>	304
5.85.3.20	<a href="#">AK_on_observable_notify</a>	305
5.85.3.21	<a href="#">AK_on_transaction_end</a>	305
5.85.3.22	<a href="#">AK_release_locks</a>	305
5.85.3.23	<a href="#">AK_remove_transaction_thread</a>	305

---

5.85.3.24 AK_search_empty_link_for_hook . . . . .	306
5.85.3.25 AK_search_existing_link_for_hook . . . . .	306
5.85.3.26 AK_search_lock_entry_list_by_key . . . . .	306
5.85.3.27 AK_transaction_finished . . . . .	306
5.85.3.28 AK_transaction_manager . . . . .	307
5.85.3.29 AK_transaction_register_observer . . . . .	307
5.85.3.30 AK_transaction_unregister_observer . . . . .	307
5.85.3.31 handle_transaction_notify . . . . .	307
 Index	 309



# 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.

#### Parameters

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

#### Returns

OK or NOT\_OK based on the success of the function.

### 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.

#### Parameters

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

#### Returns

OK or NOT\_OK based on the success of the function.

### Member [AK\\_archive\\_log](#) (int sig)

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

#### Parameters

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	associated with the transaction

#### Returns

ABORT or COMMIT based on the success of the function.

### Member [AK\\_execute\\_commands](#) (command \*, int)

Check multithreading, check if it's working correctly

## Parameters

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	associated with the transaction

## Returns

ABORT or COMMIT based on the success of the function.

**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.

## Returns

char array in format day.month.year-hour:min:sec.usecu.log

**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.

## Parameters

<i>blockMemory-Address</i>	integer representation of memory address, the hash value is calculated from this parameter.
----------------------------	---

## Returns

integer containing the hash value of the passed memory address

**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.

## Parameters

<i>blockMemory-Address</i>	integer representation of memory address, the hash value is calculated from this parameter.
----------------------------	---

## Returns

integer containing the hash value of the passed memory address



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">_file_metadata</a>	9
<a href="#">AK_agg_input</a>	9
Structure that contains attributes from table header, tasks for this table and counter value	
<a href="#">AK_agg_value</a>	10
Structure that contains attribute name, date and aggregation task associated	
<a href="#">AK_block</a>	10
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	
<a href="#">AK_block_activity</a>	11
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 un-interruptable, 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	
<a href="#">AK_blocktable</a>	12
<a href="#">AK_command_recovery_struct</a>	12
Recovery structure used to recover commands from binary file	
<a href="#">AK_command_struct</a>	12
<a href="#">AK_create_table_struct</a>	13
<a href="#">AK_db_cache</a>	13
Structure that defines global cache memory	
<a href="#">AK_header</a>	13
Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code	
<a href="#">AK_mem_block</a>	14
Structure that defines a block of data in memory	
<a href="#">AK_operand</a>	15
<a href="#">AK_query_mem</a>	15
Structure that defines global query memory	
<a href="#">AK_query_mem_dict</a>	15
Structure that defines global query memory for data dictionaries	

<a href="#">AK_query_mem_lib</a>	Structure that defines global query memory for libraries . . . . .	16
<a href="#">AK_query_mem_result</a>	Structure that defines global query memory for results . . . . .	16
<a href="#">AK_redo_log</a>	Structure that defines global redo log . . . . .	17
<a href="#">AK_ref_item</a>	Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference . . . . .	17
<a href="#">AK_results</a>	Structure used for in-memory result caching . . . . .	18
<a href="#">AK_tuple_dict</a>	Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size . . . . .	18
<a href="#">blocktable</a>	Structure that defines bit status of blocks, last initialized and last allocated index . . . . .	19
<a href="#">btree_node</a>	. . . . .	19
<a href="#">bucket_elem</a>	Structure for defining a single bucket element . . . . .	20
<a href="#">cost_eval_t</a>	Structure for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name) . . . . .	20
<a href="#">drop_arguments</a>	. . . . .	21
<a href="#">hash_bucket</a>	Structure for hash bucket for table hashing . . . . .	21
<a href="#">hash_info</a>	Structure for defining a hash info element . . . . .	21
<a href="#">intersect_attr</a>	Structure defines intersect attribute . . . . .	22
<a href="#">list_structure_ad</a>	. . . . .	22
<a href="#">list_structure_add</a>	Structure that defines linked list node for index . . . . .	23
<a href="#">main_bucket</a>	Structure for defining main bucket for table hashing . . . . .	23
<a href="#">memoryAddresses</a>	Structure that represents a linked list of locked addresses . . . . .	23
<a href="#">observable_transaction</a>	Structure which defines transaction observable type . . . . .	24
<a href="#">observable_transaction_struct</a>	. . . . .	24
<a href="#">observer_lock</a>	Structure which defines transaction lock observer type . . . . .	24
<a href="#">root_info</a>	. . . . .	25
<a href="#">search_params</a>	Structure that contains attribute name, lower and upper data value, special(NULL or *) which is input for AK_equisearch_unsorted and AK_rangesearch_unsorted . . . . .	25
<a href="#">search_result</a>	Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted . . . . .	26
<a href="#">struct_add</a>	Structure defining node address . . . . .	26
<a href="#">table_addresses</a>	Structure that defines start and end address of extent . . . . .	27
<a href="#">threadContainer</a>	Structure that represents a linked list of threads . . . . .	27
<a href="#">transaction_list_elem</a>	Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table . . . . .	28

---

<a href="#">transaction_list_head</a>	
Structure that represents LockTable entry about doubly linked list of collision in Hash table . . .	28
<a href="#">transaction_locks_list_elem</a>	
Structure that represents LockTable entry about transaction resource lock . . . . .	29
<a href="#">transactionData</a>	
Structure used to transport transaction data to the thread . . . . .	29



## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">dm/dbman.c</a>	31
<a href="#">dm/dbman.h</a>	44
<a href="#">file/blobs.c</a>	59
<a href="#">file/blobs.h</a>	62
<a href="#">file/fileio.c</a>	65
<a href="#">file/fileio.h</a>	70
<a href="#">file/files.c</a>	74
<a href="#">file/files.h</a>	75
<a href="#">file/filesearch.c</a>	77
<a href="#">file/filesearch.h</a>	78
<a href="#">file/filesort.h</a>	80
<a href="#">file/id.c</a>	82
<a href="#">file/id.h</a>	83
<a href="#">file/table.c</a>	116
<a href="#">file/table.h</a>	125
<a href="#">file/test.c</a>	134
<a href="#">file/test.h</a>	137
<a href="#">file/idx/bitmap.c</a>	84
<a href="#">file/idx/bitmap.h</a>	88
<a href="#">file/idx/btree.c</a>	93
<a href="#">file/idx/btree.h</a>	94
<a href="#">file/idx/hash.c</a>	96
<a href="#">file/idx/hash.h</a>	100
<a href="#">file/idx/index.c</a>	105
<a href="#">file/idx/index.h</a>	111
<a href="#">mm/memoman.c</a>	140
<a href="#">mm/memoman.h</a>	146
<a href="#">opti/query_optimization.c</a>	152
<a href="#">opti/query_optimization.h</a>	154
<a href="#">opti/rel_eq_assoc.c</a>	156
<a href="#">opti/rel_eq_assoc.h</a>	158
<a href="#">opti/rel_eq_comut.c</a>	159
<a href="#">opti/rel_eq_comut.h</a>	161
<a href="#">opti/rel_eq_projection.c</a>	163
<a href="#">opti/rel_eq_projection.h</a>	167
<a href="#">opti/rel_eq_selection.c</a>	171
<a href="#">opti/rel_eq_selection.h</a>	175
<a href="#">rec/archive_log.h</a>	179

rec/recovery.c	180
rec/recovery.h	??
rec/redo_log.c	182
rec/redo_log.h	??
rel/aggregation.c	183
rel/aggregation.h	187
rel/difference.c	190
rel/difference.h	191
rel/expression_check.c	192
rel/expression_check.h	195
rel/intersect.c	196
rel/intersect.h	197
rel/nat_join.c	198
rel/nat_join.h	201
rel/product.c	203
rel/product.h	204
rel/projection.c	205
rel/projection.h	210
rel/selection.c	213
rel/selection.h	215
rel/sequence.c	216
rel/sequence.h	219
rel/theta_join.c	222
rel/theta_join.h	224
rel/union.c	226
rel/union.h	227
sql/command.h	??
sql/drop.c	254
sql/drop.h	257
sql/function.c	258
sql/function.h	262
sql/privileges.c	266
sql/privileges.h	??
sql/select.c	274
sql/select.h	??
sql/trigger.c	275
sql/trigger.h	279
sql/view.c	282
sql/view.h	??
sql/cs/between.c	228
sql/cs/between.h	230
sql/cs/check_constraint.c	232
sql/cs/check_constraint.h	234
sql/cs/constraint_names.c	236
sql/cs/constraint_names.h	237
sql/cs/nnnull.c	238
sql/cs/nnnull.h	239
sql/cs/reference.c	241
sql/cs/reference.h	244
sql/cs/unique.c	251
sql/cs/unique.h	253
trans/transaction.c	285
trans/transaction.h	296

## 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 attribute 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 attribute 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 uninterruptable, 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 uninterruptable, 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** [MAX\_VARCHAR\_LENGTH]
- char **arguments** [MAX\_ATTRIBUTES][MAX\_VARCHAR\_LENGTH]
- int **finished**

### 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 constraints*

- char [constr\\_name](#) [MAX\_CONSTRAINTS][MAX\_CONSTR\_NAME]  
*extra integrity constraint names*
- char [constr\\_code](#) [MAX\_CONSTRAINTS][MAX\_CONSTR\_CODE]  
*extra integrity constraint 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\_operand Struct Reference

### Public Attributes

- char **value** [MAX\_VARCHAR\_LENGTH]
- int **type**

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

- rel/[projection.h](#)

## 4.14 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*

### 4.14.1 Detailed Description

Structure that defines global query memory.

Author

Unknown

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

- mm/[memoman.h](#)

## 4.15 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)*

### 4.15.1 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](#)

## 4.16 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)*

### 4.16.1 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](#)

## 4.17 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)*

### 4.17.1 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](#)

## 4.18 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** [MAX\_REDO\_LOG\_ENTRIES]
- int **number**

### 4.18.1 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](#)

## 4.19 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**

#### 4.19.1 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](#)

### 4.20 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]

#### 4.20.1 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](#)

### 4.21 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( \*\*\* ) )*

### 4.21.1 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](#)

## 4.22 blocktable Struct Reference

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

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

### 4.22.1 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](#)

## 4.23 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](#)

## 4.24 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*

### 4.24.1 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](#)

## 4.25 cost\_eval\_t Struct Reference

Structure 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]

### 4.25.1 Detailed Description

Structure 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](#)

## 4.26 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](#)

## 4.27 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*

### 4.27.1 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](#)

## 4.28 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*

### 4.28.1 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](#)

## 4.29 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*

### 4.29.1 Detailed Description

Structure defines intersect attribute.

Author

Dino Laktašić

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

- rel/[intersect.h](#)

## 4.30 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](#)

## 4.31 list\_structure\_add Struct Reference

Structure that defines linked list node for index.

```
#include <index.h>
```

### 4.31.1 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](#)

## 4.32 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*

### 4.32.1 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](#)

## 4.33 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**

### 4.33.1 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](#)

## 4.34 observable\_transaction Struct Reference

Structure which defines transaction observable type.

```
#include <transaction.h>
```

### 4.34.1 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](#)

## 4.35 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](#)

## 4.36 observer\_lock Struct Reference

Structure which defines transaction lock observer type.

```
#include <transaction.h>
```

## Public Attributes

- AK\_observer \* **observer**

### 4.36.1 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](#)

## 4.37 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](#)

## 4.38 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*

### 4.38.1 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](#)

## 4.39 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*

### 4.39.1 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](#)

## 4.40 struct\_add Struct Reference

Structure defining node address.

```
#include <index.h>
```

**Public Attributes**

- int [addBlock](#)  
*block address*
- int [indexTd](#)  
*index table destination*



#### 4.40.1 Detailed Description

Structure defining node address.

##### Author

Unknown

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

- file/idx/[index.h](#)

### 4.41 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 addresses*
- int **address\_to** [MAX\_EXTENTS\_IN\_SEGMENT]

#### 4.41.1 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](#)

### 4.42 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**

### 4.42.1 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](#)

## 4.43 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**

### 4.43.1 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](#)

## 4.44 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**

#### 4.44.1 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](#)

### 4.45 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**

#### 4.45.1 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](#)

### 4.46 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**

#### 4.46.1 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:

#### Functions

- 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.*
- 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.*
- 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\\_allocate\\_block\\_activity\\_modes](#) ()  
*Allocation of array which will contain information about which blocks are being accessed. Creates an array. Each element of this array will correspond to one initialized block. For more info, see explanation in [dbman.h](#).*
- 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.*
- `AK_block * AK_init_block ()`  
*Function that initializes new block.*
- `int AK_print_block (AK_block *block, int num, char *gg, FILE *fpp)`  
*Function that dumps block.*
- `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_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_copy_header (AK_header *header, int *blocknum, int num)`  
*Function copy header to blocks. Completely thread-safe.*
- `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 allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.*
- `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 allocates new blocks for increasing extent size.*
- `int AK_new_extent (int start_address, int old_size, int extent_type, AK_header *header)`  
*Function allocates 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 *constr_code)`  
*Function for creating header and initialize 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](#) ()
- void [AK\\_allocationbit\\_test](#) ()
- void [AK\\_allocationtable\\_test](#) ()

## Variables

- pthread\_mutex\_t **fileLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- char [test\\_lastCharacterWritten](#) = '\0'

This variable is used only when TEST\_MODE is ON! It is used only for testing functionality of [AK\\_thread\\_safe\\_block\\_access\\_test\(\)](#) function. It will contain first character of last written block. When reading thread reads the block (written by some other thread), it will compare the first character from this block to character contained in this variables. If they don't match, then the error occurred! It is assumed that the same block is being written to and read from (just like [AK\\_thread\\_safe\\_block\\_access\\_test](#) function works!)

- int [test\\_threadSafeBlockAccessSucceeded](#) = 1

Used in combination with [test\\_lastCharacterWritten](#). Will give the answer to question: "Has [AK\\_thread\\_safe\\_block\\_access\\_test](#) succeeded?" 0 means NO, 1 means YES.

### 5.1.1 Detailed Description

Defines functions for the disk manager

### 5.1.2 Function Documentation

#### 5.1.2.1 void [AK\\_allocate\\_block\\_activity\\_modes](#) ( )

Allocation of array which will contain information about which blocks are being accessed. Creates an array. Each element of this array will correspond to one initialized block. For more info, see explanation in [dbman.h](#).

#### Author

Domagoj Šitum

#### 5.1.2.2 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

#### 5.1.2.3 void AK\_allocationtable\_dump ( int zz )

Function dumpes allocation table.

##### Author

dv

##### Returns

nothing

#### 5.1.2.4 void AK\_blocktable\_dump ( int zz )

Function dumpes allocation table.

##### Author

dv

##### Returns

nothing

#### 5.1.2.5 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

#### 5.1.2.6 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

#### 5.1.2.7 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 Laktašić (fixed header BUG), refurbished by dv



## Parameters

<i>header</i>	pointer to header provided for copy
<i>blocknum</i>	pointer to addresses of blocks that header needs to be copied
<i>num</i>	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

#### 5.1.2.8 AK\_header\* AK\_create\_header ( char \* *name*, int *type*, int *integrity*, char \* *constr\_name*, char \* *contr\_code* )

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

## Author

Matija Novak

## Parameters

<i>name</i>	name of the attribute
<i>type</i>	type of the attribute
<i>integrity</i>	standard integrity constraint
<i>constr_name</i>	extra integrity constraint name
<i>contr_code</i>	extra integrity constraint code

## Returns

[AK\\_header](#)

initialize catalog\_header->integrity and catalog\_header->constr\_name[][] and catalog\_header->constr\_code[][] with data given as functions parameters

#### 5.1.2.9 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

<i>address</i>	address of the block to be deleted
----------------	------------------------------------

## Returns

returns EXIT\_SUCCESS if deletion successful, else EXIT\_ERROR

### 5.1.2.10 int AK\_delete\_extent ( int *begin*, int *end* )

Function deletes an extent between begin and end blocks.

#### Author

Dejan Samboliæ

#### Parameters

<i>begin</i>	address of extent's first block
<i>end</i>	address of extent's last block

#### Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 5.1.2.11 int AK\_delete\_segment ( char \* *name*, int *type* )

#### Author

Mislav Ėakariæ

#### Parameters

<i>name</i>	name of the segment
<i>type</i>	type of the segment

#### Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 5.1.2.12 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

<i>bitset</i>	int pointer, cointainer for bit set
<i>fromWhere</i>	has meaning just in SEQUENCE case. It describes from which address searching have to start.
<i>gaplength</i>	tells how many used blocks could be tolerated in bitset
<i>num</i>	Tells how many AK_free blocks has been needed
<i>mode</i>	Defines how to obtain set of indexes to AK_free addresses
<i>target</i>	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.

5.1.2.13 `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 allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

Author

dv

Parameters

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

Returns

pointer to set of allocated block addresses

vars for loop [for]

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

5.1.2.14 `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 allocates new blocks for increasing extent size.

Author

dv

Parameters

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

Returns

pointer to set of allocated block addresses

vars for loop [for]

5.1.2.15 `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

**5.1.2.16 AK\_block\* AK\_init\_block ( )**

Function that initializes new block.

**Author**

Markus Schatten , rearranged by dv

**Returns**

pointer to block allocated in memory

**5.1.2.17 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**

<i>size</i>	size of new file in in blocks
-------------	-------------------------------

**Returns**

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

**5.1.2.18 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.

## 5.1.2.19 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

5.1.2.20 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

<i>relation</i>	address of system table of relation in db_file
<i>attribute</i>	address of system table of attribute in db_file
<i>index</i>	address of system table of index in db_file
<i>view</i>	address of system table of view in db_file
<i>sequence</i>	address of system table of sequence in db_file
<i>function</i>	address of system table of function in db_file
<i>function_ - arguments</i>	address of system table of function_arguments in db_file
<i>trigger</i>	address of system table of trigger in db_file
<i>trigger_ - conditions</i>	address of system table of trigger_conditions in db_file
<i>db</i>	address of system table of db in db_file
<i>db_obj</i>	address of system table of db_obj in db_file
<i>user</i>	address of system table of user in db_file
<i>group</i>	address of system table of group in db_file
<i>user_group</i>	address of system table of users associated with groups in db_file
<i>user_right</i>	address of system table of user right in db_file
<i>group_right</i>	address of system table of group right in db_file
<i>constraint</i>	address of system table of constraint in db_file
<i>constraintNull</i>	address of system table of constraintNull in db_file
<i>constraintCheck</i>	system table address for check constraint
<i>reference</i>	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)

#### 5.1.2.21 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

<i>block_address</i>	adress of a block in which we want insert data
<i>type</i>	type of entry_data
<i>entry_data</i>	(char) data which is inserted, can be int but must first be converted to char
<i>i</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 original block

using strlen becuse sizeof(entry\_data) is always 4 copy data into bloc->data on start position bloc->AK\_free\_space

adress 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

#### 5.1.2.22 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

<i>block</i>	pointer to the block of memory to fill
<i>value</i>	int value to be set
<i>num</i>	number of ints in the block of memory to be set

##### Returns

No return value

### 5.1.2.23 int AK\_new\_extent ( int *start\_address*, int *old\_size*, int *extent\_type*, AK\_header \* *header* )

Function allocates 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 Laktašić (fixed header BUG), refurbished by dv

#### Parameters

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>old_size</i>	size of previous extent in same segment (in blocks)
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	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 successfully allocated, which means that the extend allocation has FAILED

### 5.1.2.24 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

<i>name</i>	(character pointer) name of segment
<i>type</i>	segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	(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

### 5.1.2.25 int AK\_print\_block ( AK\_block \* *block*, int *num*, char \* *gg*, FILE \* *fpp* )

Function that dumps block.

**Author**

dv

**Returns**

nothing

**5.1.2.26 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**

<i>address</i>	block number (address)
----------------	------------------------

**Returns**

pointer to block allocated in memory

**5.1.2.27 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

**5.1.2.28 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

<i>relation</i>	relation in database
<i>attribute</i>	attribute in database
<i>index</i>	index in database
<i>view</i>	view in database
<i>sequence</i>	sequence in database
<i>function</i>	function in database
<i>function_ - arguments</i>	functional_arguments in database
<i>trigger</i>	trigger in database
<i>trigger_ - conditions</i>	trigger conditions in database
<i>db</i>	database
<i>db_obj</i>	database object
<i>user</i>	user in database
<i>group</i>	group in database
<i>user_group</i>	user associated with group in database
<i>user_right</i>	user right in database
<i>group_right</i>	group right in database
<i>constraint</i>	constraint in database
<i>constraintNull</i>	Null constraint in database
<i>constraintCheck</i>	Check constraint in database
<i>reference</i>	reference database

## Returns

EXIT\_SUCCESS

## 5.1.2.29 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

## 5.1.2.30 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**

<i>block</i>	pointer to block allocated in memory to write
--------------	---

**Returns**

EXIT\_SUCCESS if successful, EXIT\_ERROR otherwise

**5.1.2.31 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

**5.1.2.32 int fsize ( FILE \* *fp* )**

Helper function to determine file size.

**Returns**

file size

**5.2 dm/dbman.h File Reference**

```
#include "../auxiliary/auxiliary.h"
#include <errno.h>
#include <pthread.h>
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../auxiliary/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 uninterruptable, 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 200`
- `#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 of allocation table LOWER - set tries to place itself to lower part of 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 allocates 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 allocates 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 allocates 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 \*constr\_code)

*Function for creating header and initialize 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 attribtues 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 of allocation table LOWER - set tries to place itself to lower part of 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 dumps 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 Laktašić (fixed header BUG), refurbished by dv

##### Parameters

<i>header</i>	pointer to header provided for copy
<i>blocknum</i>	pointer to addresses of blocks that header needs to be copied
<i>num</i>	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 initialize integrity, constraint name and constraint code with parameter values of function.

**Author**

Matija Novak

**Parameters**

<i>name</i>	name of the attribute
<i>type</i>	type of the attribute
<i>integrity</i>	standard integrity constraint
<i>constr_name</i>	extra integrity constraint name
<i>contr_code</i>	extra integrity constraint 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**

<i>address</i>	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 Samboljæ

**Parameters**

<i>begin</i>	address of extent's first block
<i>end</i>	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 Ėakariæ

## Parameters

<i>name</i>	name of the segment
<i>type</i>	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

<i>bitset</i>	int pointer, cointainer for bit set
<i>fromWhere</i>	has meaning just in SEQUENCE case. It describes from which address searching have to start.
<i>gaplength</i>	tells how many used blocks could be tolerated in bitset
<i>num</i>	Tells how many AK_free blocks has been needed
<i>mode</i>	Defines how to obtain set of indexes to AK_free addresses
<i>target</i>	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 allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

## Author

dv

## Parameters

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>desired_size</i>	number of desired blocks

<i>AK_allocation_ - set_mode</i>	a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
<i>border</i>	number of allocated blocks gap
<i>target</i>	block address around which other blocks have to be searched
<i>header</i>	pointer to header that should be written to the new extent (all blocks)
<i>int</i>	gl gap size

**Returns**

pointer to set of allocated block addresses

vars for loop [for]

if some blocks are not successfully 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 allocates new blocks for increasing extent size.

**Author**

dv

**Parameters**

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

**Returns**

pointer to set of allocated 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

<i>size</i>	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

<i>relation</i>	address of system table of relation in db_file
<i>attribute</i>	address of system table of attribute in db_file
<i>index</i>	address of system table of index in db_file
<i>view</i>	address of system table of view in db_file
<i>sequence</i>	address of system table of sequence in db_file
<i>function</i>	address of system table of function in db_file
<i>function_ - arguments</i>	address of system table of function_arguments in db_file
<i>trigger</i>	address of system table of trigger in db_file
<i>trigger_ - conditions</i>	address of system table of trigger_conditions in db_file
<i>db</i>	address of system table of db in db_file
<i>db_obj</i>	address of system table of db_obj in db_file
<i>user</i>	address of system table of user in db_file
<i>group</i>	address of system table of group in db_file
<i>user_group</i>	address of system table of users associated with groups in db_file
<i>user_right</i>	address of system table of user right in db_file
<i>group_right</i>	address of system table of group right in db_file
<i>constraint</i>	address of system table of constraint in db_file
<i>constraintNull</i>	address of system table of constraintNull in db_file
<i>constraintCheck</i>	system table address for check constraint
<i>reference</i>	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**

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

**Returns**

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

<i>block</i>	pointer to the block of memory to fill
<i>value</i>	int value to be set
<i>num</i>	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 allocates 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 Laktašić (fixed header BUG), refurbished by dv

## Parameters

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>old_size</i>	size of previous extent in same segment (in blocks)
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	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 successfully 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

<i>name</i>	(character pointer) name of segment
<i>type</i>	segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	(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**

<i>address</i>	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**

<i>relation</i>	relation in database
<i>attribute</i>	attribute in database
<i>index</i>	index in database
<i>view</i>	view in database
<i>sequence</i>	sequence in database
<i>function</i>	function in database
<i>function_ - arguments</i>	functional_arguments in database
<i>trigger</i>	trigger in database
<i>trigger_ - conditions</i>	trigger conditions in database

<i>db</i>	database
<i>db_obj</i>	database object
<i>user</i>	user in database
<i>group</i>	group in database
<i>user_group</i>	user associated with group in database
<i>user_right</i>	user right in database
<i>group_right</i>	group right in database
<i>constraint</i>	constraint in database
<i>constraintNull</i>	Null constraint in database
<i>constraintCheck</i>	Check constraint in database
<i>reference</i>	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**

<i>block</i>	pointer 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 splitting 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 globally 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\_io\_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 splitting 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 splitting 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 globally 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 splitting 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

<i>row_root</i>	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

<i>id</i>	id of row
<i>tableName</i>	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

<i>temp_block</i>	block to work with
<i>row_list</i>	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

<i>row_root</i>	elements of one row
<i>del</i>	- 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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

<i>row_root</i>	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 whether block contains 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**

<i>row_root</i>	list of elements to insert
<i>temp_block</i>	block in which we insert data

**Returns**

EXIT\_SUCCESS 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**

<i>row_root</i>	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**

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

**Returns**

No return value

**5.6 file/fileio.h File Reference**

```
#include "../auxi/constants.h"
#include "../sql/cs/reference.h"
#include "../mm/memoman.h"
#include "../rec/recovery.h"
#include "../rec/archive_log.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 elements 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 whether block contains 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 violate 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 are 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

<i>row_root</i>	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

<i>id</i>	id of row
<i>tableName</i>	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

<i>temp_block</i>	block to work with
<i>row_list</i>	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

<i>row_root</i>	elements of one row
<i>del</i>	- 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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 elements are set according to function arguments. Pointers are changed so that before element points to new element.

## Author

Matija Novak

## Parameters

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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 violate 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

<i>row_root</i>	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 whether block contains 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

<i>row_root</i>	list of elements to insert
<i>temp_block</i>	block in which we insert data

##### Returns

EXIT\_SUCCESS 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

<i>row_root</i>	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

<i>temp_block</i>	block to work with
<i>row_list</i>	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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	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\_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.*
- 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

<i>srResult</i>	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

<code>szRelation</code>	relation name
<code>aspParams</code>	array of search parameters
<code>iNum_search_params</code>	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\_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.*

- 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

<i>srResult</i>	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

<code>szRelation</code>	relation name
<code>aspParams</code>	array of search parameters
<code>iNum_search_params</code>	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`  
*Constant declaring size of data to be compared.*
- `#define DATA_TUPLE_SIZE 500`  
*Constant 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

<i>iBlock</i>	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 return 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 return 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 numAttributes, [AK\\_header](#) \*headerIndex)  
*Function that loads index table with value of particular attribute.*
- list\_ad \* [Ak\\_get\\_Attribute](#) (char \*indexName, char \*attribute)  
*Function gets addresses of the particular 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 addresses.*
- 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 already 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 recreates 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 recreates index values again if different number of params is detected.

##### Author

Lovro Predovan

##### Parameters

<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>newAttribute-Value</i>	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 *numAttributes*, AK\_header \* *headerIndex* )

Function that loads index table with value of particular attribute.

##### Author

Saša Vukšić, Lovro Predovan

##### Parameters

<i>tblName</i>	source table
----------------	--------------

<i>tblNameIndex</i>	new name of index table
<i>attributeName</i>	attribute on which we make index
<i>positionTbl</i>	position of attribute in header of table
<i>numAtributes</i>	number of attributes in table
<i>headerIndex</i>	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**

<i>tblName</i>	name of table
<i>attributes</i>	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**

<i>Bitmap</i>	index table name
---------------	------------------

**Returns**

No return value

**5.14.2.6 list\_ad\* Ak\_get\_Attribute ( char \* *indexName*, char \* *attribute* )**

Function gets addresses 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

<i>indexName</i>	name of index
<i>attribute</i>	name of attribute

## Returns

list of addresses

**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

<i>tableName</i>	name of table
<i>attributeValue</i>	value of attribute

## Returns

list of addresses

**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

<i>L</i>	list of elements
<i>ele</i>	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 addresses.

## Author

Saša Vukšić, Lovro Predovan

## Parameters

<i>list</i>	list of addresses
-------------	-------------------

## 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

<i>tblName</i>	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 already 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

<i>addBlock</i>	adress of block
<i>addTD</i>	adress of tuple dict
<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>attributeValue</i>	value of attribute
<i>newAttribute-Value</i>	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

<i>block</i>	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 num-Attributes, [AK\\_header](#) \*headerIndex)  
*Function that loads index table with value of particular attribute.*
- [list\\_ad](#) \* [Ak\\_get\\_Attribute](#) (char \*indexName, char \*attribute)  
*Function gets adresses of the particulary 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 addresses.*
- [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 alreedy 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 recreates index values again if different number of params is detected.

##### Author

Lovro Predovan

##### Parameters

<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>newAttribute-Value</i>	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 *numAttributes*, AK\_header \* *headerIndex* )

Function that loads index table with value of particular attribute.

##### Author

Saša Vukšić, Lovro Predovan

##### Parameters

<i>tblName</i>	source table
<i>tblNameIndex</i>	new name of index table
<i>attributeName</i>	attribute on which we make index
<i>positionTbl</i>	position of attribute in header of table
<i>numAttributes</i>	number of attributes in table
<i>headerIndex</i>	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**

<i>tblName</i>	name of table
<i>attributes</i>	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**

<i>Bitmap</i>	index table name
---------------	------------------

**Returns**

No return value

**5.15.2.6 list\_ad\* Ak\_get\_Attribute ( char \* *indexName*, char \* *attribute* )**

Function gets addresses 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**

<i>indexName</i>	name of index
<i>attribute</i>	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

<i>tableName</i>	name of table
<i>attributeValue</i>	value of attribute

##### Returns

list of addresses

#### 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

<i>L</i>	list of elements
<i>ele</i>	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 addresses.

##### Author

Saša Vukšić, Lovro Predovan

##### Parameters

<i>list</i>	list of addresses
-------------	-------------------

##### 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

<i>tblName</i>	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 already 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

<i>addBlock</i>	adress of block
<i>addTD</i>	adress of tuple dict
<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>attributeValue</i>	value of atribute
<i>newAttribute-Value</i>	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

<i>block</i>	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

Andelko Spevec

##### Parameters

<i>tblName</i>	- name of the table on which we are creating index
<i>attributes</i>	- attribute on which we are creating index
<i>indexName</i>	- 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

Andelko Spevec

##### Parameters

<i>indexName</i>	- name of the index
<i>searchValue</i>	- value that we are searching in the index
<i>endRange</i>	- if 0 search is for 0 value, else searching in range
<i>toDo</i>	- 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 "../aux/constants.h"
#include "../aux/configuration.h"
#include "../aux/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 structures 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

Andelko Spevec

#### Parameters

<i>tblName</i>	- name of the table on which we are creating index
<i>attributes</i>	- attribute on which we are creating index
<i>indexName</i>	- 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

Andelko Spevec

## Parameters

<i>indexName</i>	- name of the index
<i>searchValue</i>	- value that we are searching in the index
<i>endRange</i>	- if 0 search is for 0 value, else searching in range
<i>toDo</i>	- 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**

<i>indexName</i>	name of index
<i>modulo</i>	value for modulo hash function
<i>main_bucket_num</i>	number of main buckets
<i>hash_bucket_num</i>	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**

<i>tblName</i>	name of table for which the index is being created
<i>indexName</i>	name of index
<i>attributes</i>	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**

<i>indexName</i>	name of index
<i>values</i>	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

<i>elem</i>	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

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index
<i>delete</i>	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

<i>indexName</i>	name of index
<i>values</i>	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

<i>indexName</i>	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

<i>indexName</i>	name of index
<i>n</i>	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

<i>indexName</i>	name of index
<i>data</i>	content of bucket stored in char array
<i>type</i>	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

<i>indexName</i>	name of index
<i>hashValue</i>	hash value of record that is being inserted
<i>add</i>	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

<i>add</i>	address of where the bucket is stored
<i>data</i>	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 "../aux/constants.h"
#include "../aux/configuration.h"
#include "../files.h"
#include "../aux/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

<i>indexName</i>	name of index
<i>modulo</i>	value for modulo hash function
<i>main_bucket_num</i>	number of main buckets

<i>hash_bucket_ - num</i>	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**

<i>tblName</i>	name of table for which the index is being created
<i>indexName</i>	name of index
<i>attributes</i>	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**

<i>indexName</i>	name of index
<i>values</i>	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

<i>elem</i>	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

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index
<i>delete</i>	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

<i>indexName</i>	name of index
<i>values</i>	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

<i>indexName</i>	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

<i>indexName</i>	name of index
<i>n</i>	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

<i>indexName</i>	name of index
<i>data</i>	content of bucket stored in char array
<i>type</i>	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

<i>indexName</i>	name of index
<i>hashValue</i>	hash value of record that is being inserted
<i>add</i>	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

<i>add</i>	address of where the bucket is stored
<i>data</i>	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 "../aux/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 initializing 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

<a href="#">L</a>	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

<i>Deletedelement- _op</i>	- address of node to delete
<i>list_ad</i>	*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

<i>list_ad</i>	*L linked list head
----------------	---------------------

## Returns

Address of first node

**5.20.2.4 AK\_header\* AK\_get\_index\_header ( char \* indexTblName )**

Function that gets 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

<i>*tblName</i>	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**

<i>*tableName</i>	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**

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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**

<i>list_ad</i>	*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**

<i>Currentelement- _op</i>	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**

<i>Searchedelement- _op</i>	address of current note
<i>*L</i>	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**

<i>Currentelement- _op</i>	Address of current node
<i>L</i>	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

<i>tblName</i>	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

<i>list_ad</i>	*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

<i>addBlock</i>	address block
<i>indexTd</i>	index table destination
<i>*attname</i>	attribute name
<i>elementBefore</i>	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

<i>index</i>	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

<i>*tblName</i>	table name
-----------------	------------

## Returns

No return value

## 5.21 file/idx/index.h File Reference

```
#include "../aux/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 initializing 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

<i>L</i>	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

<i>Deletedelement- _op</i>	- address of node to delete
<i>list_ad</i>	*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

<i>list_ad</i>	*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

<i>*tableName</i>	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

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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

<i>list_ad</i>	*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

<i>Currentelement-op</i>	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**

<i>Searchedelement- _op</i>	address of current note
<i>*L</i>	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**

<i>Currentelement- _op</i>	Address of current node
<i>L</i>	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**

<i>tblName</i>	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

<i>list_ad</i>	*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

<i>addBlock</i>	address block
<i>indexTd</i>	index table destination
<i>*attname</i>	attribute name
<i>elementBefore</i>	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

<i>index</i>	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

<i>*tblName</i>	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 gets 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 attribute.*
- 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

<a href="#">tbl1_temp_block</a>	first cache block of the first table
<a href="#">tbl2_temp_block</a>	first cache block of the second table
<a href="#">operator_name</a>	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 attribute.

Author

Matija Šestak.

Parameters

<i>*tblName</i>	table name
<i>*attrName</i>	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

<i>*tblName</i>	table name
<i>index</i>	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

<i>num</i>	zero-based column index
<i>*tblName</i>	table name

Returns

column values list

#### 5.22.2.5 AK\_header\* AK\_get\_header ( char \* *tblName* )

Function that gets 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**

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>num</i>	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

<i>*table</i>	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

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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**

<i>col_len</i> []	array of max lengths for each attribute
* <i>row</i>	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**

<i>col_len</i> []	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len</i> []	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len</i> []	array of max lengths for each attribute
* <i>row</i>	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**

* <i>tblName</i>	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

<i>*tblName</i>	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

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	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

<i>*tblName</i>	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

<i>tblName</i>	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

<i>*tuple</i>	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 Palać

## Parameters

<i>column</i>	index of column attribute
<i>*row</i>	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 gets 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 attribute.*
- 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](#) (int column, struct list\_node \*node)
  - Function that returns value of attribute from row.*
- 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

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

### 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

<i>tbl1_temp_block</i>	first cache block of the first table
<i>tbl2_temp_block</i>	first cache block of the second table
<i>operator_name</i>	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 attribute.

**Author**

Matija Šestak.

**Parameters**

<i>*tblName</i>	table name
<i>*attrName</i>	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**

<i>*tblName</i>	table name
<i>index</i>	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**

<i>num</i>	zero-based column index
<i>*tblName</i>	table name

**Returns**

column values list

**5.23.2.5 AK\_header\* AK\_get\_header ( char \* tblName )**

Function that gets 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

<i>*tblName</i>	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

<i>*tableName</i>	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

<i>num</i>	zero-based row index
<i>*</i>	<i>tblName</i> 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**

<i>*table</i>	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**

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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**

<i>*</i>	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

<i>col_len</i> []	array of max lengths for each attribute
* <i>row</i>	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

<i>col_len</i> []	max lengths for each attribute cell
<i>length</i>	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

<i>col_len</i> []	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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**

<i>*tblName</i>	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**

<i>*tblName</i>	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**

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	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**

<i>*tblName</i>	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**

<i>*tuple</i>	tuple in the list
---------------	-------------------

**Returns**

tuple value as a string

### 5.23.2.22 `char* get_row_attr_data ( int column, struct list_node * node )`

Function that returns value of attribute from row.

#### Author

Leon Palać

#### Parameters

<i>column</i>	index of column attribute
<i>*row</i>	list with row elements

#### Returns

attribute data

## 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 "../auxiliary/auxiliary.h"
#include "../opti/rel_eq_comut.h"
Include dependency graph for test.c:
```

### Functions

- `char * AK_get_table_attribute_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\_attribute\_types ( char \* tblName )

returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER

#### Author

Goran Štok

#### Parameters

<i>tblName</i>	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

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>_num</i>	- number of attributes
<i>_type</i>	- 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

<i>num</i>	- 0 based index of column
<i>tbl</i>	- 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

<i>num</i>	- 0 based index of row
<i>tbl</i>	- 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

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>attr_value</i>	- values of attributes to be inserted
<i>_num</i>	- number of attributes
<i>_type</i>	- 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

<i>src_table</i>	- name of the source table •
<i>dest_table</i>	- table in which selection will be stored
<i>sel_query</i>	- array of operators, operands and attributes (postfix query)
<i>_num</i>	- number of attributes
<i>_type</i>	- 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 "../aux/mempro.h"
```

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

## Functions

- char \* [AK\\_get\\_table\\_attribute\\_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\_attribute\_types ( char \* tblName )**

returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER

**Author**

Goran Štok

**Parameters**

<i>tblName</i>	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**

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>_num</i>	- number of attributes
<i>_type</i>	- 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**

<i>num</i>	- 0 based index of column
<i>tbl</i>	- 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

<i>num</i>	- 0 based index of row
<i>tbl</i>	- 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

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>attr_value</i>	- values of attributes to be inserted
<i>_num</i>	- number of attributes
<i>_type</i>	- 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

<i>src_table</i>	- name of the source table •
------------------	---------------------------------

<i>dest_table</i>	- table in which selection will be stored
<i>sel_query</i>	- array of operators, operands and attributes (postfix query)
<i>_num</i>	- number of attributes
<i>_type</i>	- 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 getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_segment\\_addresses](#) (char \*segmentName)  
*Function for getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_table\\_addresses](#) (char \*table)  
*function for getting addresses of some table*
- [table\\_addresses](#) \* [AK\\_get\\_index\\_addresses](#) (char \*index)  
*Function for getting addresses of some index.*
- int [AK\\_find\\_AK\\_free\\_space](#) ([table\\_addresses](#) \*addresses)  
*Function to find AK\_free space in some block between 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

<i>num</i>	block number (address)
<i>mem_block</i>	address of memory 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 between block addresses. It's made for insert\_row()

##### Author

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

##### Parameters

<i>address</i>	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

<i>num</i>	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

<i>index</i>	index name that you search for
--------------	--------------------------------

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses 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

<i>table</i>	table name that you search for
--------------	--------------------------------

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses 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 getting addresses of some table.

##### Author

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

##### Parameters

<i>table</i>	table name that you search for
--------------	--------------------------------

##### Returns

structure `table_addresses` witch contains start and end addresses 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 getting addresses of some table

##### Author

Mislav Čakarić

##### Parameters

<i>table</i>	table name that you search for
--------------	--------------------------------

##### Returns

structure `table_addresses` witch contains start and end addresses 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

<i>table_name</i>	name of segment to extent
<i>extent_type</i>	type of extent (can be one of: <code>SEGMENT_TYPE_SYSTEM_TABLE</code> , <code>SEGMENT_TYPE_TABLE</code> , <code>SEGMENT_TYPE_INDEX</code> , <code>SEGMENT_TYPE_TRANSACTION</code> , <code>SEGMENT_TYPE_TEMP</code> )

##### 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 "../aux/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 getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_index\\_segment\\_addresses](#) (char \*segmentName)  
*Function for getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_table\\_addresses](#) (char \*table)  
*function for getting addresses of some table*
- [table\\_addresses](#) \* [AK\\_get\\_index\\_addresses](#) (char \*index)  
*Function for getting addresses of some index.*
- int [AK\\_find\\_AK\\_free\\_space](#) ([table\\_addresses](#) \*addresses)  
*Function to find AK\_free space in some block between 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**

<i>num</i>	block number (address)
<i>mem_block</i>	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 between block addresses. It's made for insert\_row()

**Author**

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

**Parameters**

<i>address</i>	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

<i>num</i>	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 getting addresses of some index.

**Author**

Mislav Čakarić

**Parameters**

<i>index</i>	index name that you search for
--------------	--------------------------------

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses 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 getting addresses of some table.

**Author**

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

**Parameters**

<i>table</i>	table name that you search for
--------------	--------------------------------

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses 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 getting addresses of some table.

**Author**

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

**Parameters**

<i>table</i>	table name that you search for
--------------	--------------------------------

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses 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 getting addresses of some table

**Author**

Mislav Čakarić

## Parameters

<i>table</i>	table name that you search for
--------------	--------------------------------

## Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from 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

<i>table_name</i>	name of segment to extent
<i>extent_type</i>	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

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>rel_eq</i>	rel_eq to execute
<i>*FLAGS</i>	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

<i>*list_query</i>	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**

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>*FLAGS</i>	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**

<i>*list_query</i>	query to be optimized
--------------------	-----------------------

**Returns**

No return value

**5.29 opt/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

* <i>list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>rel_eq</i>	rel_eq to execute
* <i>FLAGS</i>	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

* <i>list_query</i>	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**

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>*FLAGS</i>	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**

<i>*list_query</i>	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

<i>*a</i>	first value
<i>*b</i>	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

<i>*list_rel_eq</i>	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

<i>*list_rel_eq</i>	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 "../aux/mempro.h"
#include "../aux/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](#)  
*Structure 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

<i>*a</i>	first value
<i>*b</i>	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

<i>*list_rel_eq</i>	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

<i>*list_rel_eq</i>	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

<i>*list_rel_eq</i>	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

<i>*cond</i>	condition array that contains condition data
--------------	--

<i>*tblName</i>	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**

<i>*list_rel_eq</i>	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 "../aux/mempro.h"
#include "../aux/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

<i>*list_rel_eq</i>	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

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	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

<i>*list_rel_eq</i>	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 "../auxiliary/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

<i>*list_rel_eq</i>	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

<i>list_elem_attribs</i>	list element containing projection data
<i>list_elem_conds</i>	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

<i>list_elem</i>	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

<i>*tblName</i>	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.  $p[L1...]) = p[L1](R)$  Rule 3a. distribution according to theta join, only if join includes attributes from  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

## 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 reached
8. on loop exit return EXIT\_SUCCESS

## Parameters

<i>list_elem_set</i>	first list element containing projection attributes
<i>list_elem_subset</i>	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

<i>*list_rel_eq</i>	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 \* *attrs*, 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

<i>*attrs</i>	projection attributes delimited by ";" (ATTR_DELIMITER)
<i>*tblName</i>	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 \* *attrs* )

Function which removes duplicate attributes from attributes expresion.

## Author

Dino Laktašić.

## Parameters

<i>*attrs</i>	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 "../aux/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\_attrs, 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 \*attrs, 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 \*attrs)  
*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

<i>*list_rel_eq</i>	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

<i>list_elem_attribs</i>	list element containing projection data
<i>list_elem_conds</i>	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

<i>list_elem</i>	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

<i>*tblName</i>	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.  $p[L1...]) = p[L1](R)$  Rule 3a. distribution according to theta join, only if join includes attributes from  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

## 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 reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>list_elem_set</i>	first list element containing projection attributes
<i>list_elem_subset</i>	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**

<i>*list_rel_eq</i>	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 \* *attrs*, 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**

<i>*attrs</i>	projection attributes delimited by ";" (ATTR_DELIMITER)
<i>*tblName</i>	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 \* *attrs* )

Function which removes duplicate attributes from attributes expresion.

## Author

Dino Laktašić.

## Parameters

<i>*attrs</i>	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\\_attributes\\_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 instead - 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**

<i>*list_rel_eq</i>	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**

<i>*cond</i>	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\_attributes\_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**

<i>*tblName</i>	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 reached
8. on loop exit return EXIT\_SUCCESS

## Parameters

<i>*set</i>	set array
<i>*subset</i>	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

<i>*list_rel_eq</i>	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

<i>*set</i>	first set of attributes delimited by ";" (ATTR_DELIMITER)
<i>*subset</i>	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 instead - 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 discharge 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

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	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

<i>*cond</i>	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\\_attributes\\_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 instead - 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**

<i>*list_rel_eq</i>	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**

<i>*cond</i>	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\_atrrributes\_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**

<i>*tblName</i>	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 reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>*set</i>	set array
<i>*subset</i>	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**

<i>*list_rel_eq</i>	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

<i>*set</i>	first set of attributes delimited by ";" (ATTR_DELIMITER)
<i>*subset</i>	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 instead - 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 discharge 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

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	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

<i>*cond</i>	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](#) (int sig)  
*Function for making archive log.*
- void [AK\\_empty\\_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 ( int sig )

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 return value

#### 5.38.2.2 void AK\_empty\_archive\_log ( )

Empties archive log

### 5.38.2.3 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

<i>fileName</i>	- 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

<i>sig</i>	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

<i>table</i>	- table name to insert to
<i>attributes</i>	- 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

<i>input</i>	- input to tokenize
<i>delimiter</i>	- delimiter
<i>valuesOrNot</i>	- 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\\_redolog\\_commit](#) ()
- 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\_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.*
- int [AK\\_header\\_size](#) ([AK\\_header](#) \*header)
 

*Function calculates how many 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 with 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\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning 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 assigns 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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	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 beginning of input object.

## Author

Dejan Frankovic

## Parameters

<i>input</i>	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 with init values.

## Author

Dejan Frankovic

## Parameters

<i>input</i>	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\_AVG\_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns 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

<i>input</i>	input object with list of attributes by which we aggregate and types of aggregations
<i>source_table</i>	- table name for the source table
<i>agg_table</i>	table name for aggregated table

**Returns**

EXIT\_SUCCESS if continues succesfully, 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 many attributes there are in a header with while loop.

**Author**

Dejan Frankovic

**Parameters**

<i>header</i>	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

<i>szRelation</i>	relation name
<i>aspParams</i>	array of search parameters
<i>iNum_search_params</i>	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 "../aux/mempro.h"
#include "../sql/drop.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 attribute 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 many 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 with 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\_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.*
- 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 assigns 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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	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 beginning of input object.

## Author

Dejan Frankovic

## Parameters

<i>input</i>	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 with init values.

## Author

Dejan Frankovic

## Parameters

<i>input</i>	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\_AVG\_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns 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**

<i>input</i>	input object with list of attributes by which we aggregate and types of aggregations
<i>source_table</i>	- table name for the source table
<i>agg_table</i>	table name for aggregated table

**Returns**

EXIT\_SUCCESS if continues succesfully, 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 many attributes there are in a header with while loop.

**Author**

Dejan Frankovic

**Parameters**

<i>header</i>	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. Specially 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. Specially 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 "../sql/drop.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. Specially 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. Specially 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 arithmetic 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 character wildcard (%,\_ ) ch in string s with repl characters.*
- 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 arithmetic 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

<i>el</i>	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
<i>*op</i>	comparison operator
<i>*a</i>	left operand
<i>*b</i>	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

<i>row_root</i>	beginning of the row that is to be evaluated
<i>*expr</i>	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

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression
<i>checkWildCard</i>	replaces SQL wildcard to corresponding POSIX regex character
<i>sensitive</i>	case insensitive indicator 1-case sensitive,0- case insensitive

## Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding 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

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression

## Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression

#### 5.45.2.5 char\* AK\_replace\_wild\_card ( const char \* *s*, char *ch*, const char \* *repl* )

Function that replaces character wildcard (%,\_ ) ch in string s with repl characters.

Leon Palaić

## Parameters

<i>s</i>	input string
<i>ch</i>	character to be replaced

## Returns

new sequence of characters



## 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 arithmetic 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 arithmetic 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

<i>el</i>	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
<i>*op</i>	comparison operator
<i>*a</i>	left operand
<i>*b</i>	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

<i>row_root</i>	beginning of the row that is to be evaluated
<i>*expr</i>	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

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression
<i>checkWildCard</i>	replaces SQL wildcard to corresponding POSIX regex character
<i>sensitive</i>	case insensitive indicator 1-case sensitive,0- case insensitive

##### Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding 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

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression

##### Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 "../sql/drop.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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 between 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>att</i>	attributes on which we make nat_join
<i>new_table</i>	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**

<i>table_address1</i>	address of the block of the first table
<i>table_address2</i>	address of the block of the second table
<i>new_table</i>	name of the join table
<i>att_root</i>	tributes 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 between two tables on some attributes.

**Author**

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

**Parameters**

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>att</i>	attributes on which we make nat_join
<i>dstTable</i>	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**

<i>row_root</i>	- list of values from the first table to be merged with table2
<i>row_root_insert</i>	- list of values from the first table to be inserted into nat_join table
<i>temp_block</i>	- block from the second table to be merged
<i>new_table</i>	- 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 "../sql/drop.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 between 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>att</i>	attributes on which we make nat_join
<i>new_table</i>	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

<i>table_address1</i>	address of the block of the first table
<i>table_address2</i>	address of the block of the second table
<i>new_table</i>	name of the join table
<i>att_root</i>	tributes 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 between two tables on some attributes.

## Author

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

## Parameters

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>att</i>	attributes on which we make nat_join
<i>dstTable</i>	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

<i>row_root</i>	- list of values from the first table to be marged with table2
<i>row_root_insert</i>	- list of values from the first table to be inserted into nat_join table
<i>temp_block</i>	- block from the second table to be merged
<i>new_table</i>	- 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 "../aux/mempro.h"
#include "../sql/drop.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**

<code>srcTable1</code>	name of the first table
<code>srcTable2</code>	name of the second table
<code>dstTable</code>	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.*
- char \* [AK\\_get\\_operator](#) (char \*exp)  
*Function that fetches arithmetics operator.*
- void [removeSubstring](#) (char \*s, const char \*toremove)  
*Function that removes specified part of string.*
- int [AK\\_determine\\_header\\_type](#) (int a, int b)

*Determines new header type.*

- char \* [AK\\_create\\_header\\_name](#) (char \*first, char \*second, char \*operator)

*Creates new header name from passed operand names and operator.*

- 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.*

- char \* [AK\\_perform\\_operatrion](#) (char \*op, struct [AK\\_operand](#) \*ab, struct [AK\\_operand](#) \*bb, int type)

*Performs arithmetics operation on operand data.*

- 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

<i>old_block</i>	block from which we copy data
<i>dstTable</i>	name of the new table
<i>att</i>	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

<i>old_block_add</i>	address of the block from which we copy headers we need
<i>dstTable</i>	name of the new table
<i>att</i>	list of the attributes which should the projeciton table contain

#### Returns

No return value

5.53.2.3 char\* AK\_create\_header\_name ( char \* *first*, char \* *second*, char \* *operator* )

Creates new header name from passed operand names and operator.

Leon Palaić

## Parameters

<i>first</i>	operand name
<i>second</i>	operand name
<i>operator</i>	

## Returns

new sequence of characters

5.53.2.4 int AK\_determine\_header\_type ( int *a*, int *b* )

Determines new header type.

Leon Palačić

## Parameters

<i>a</i>	operand type
<i>b</i>	operand type

## Returns

header type

5.53.2.5 char\* AK\_get\_operator ( char \* *exp* )

Function that fetches arithmetics operator.

Leon Palačić

## Parameters

<i>exp</i>	input expression string
------------	-------------------------

## Returns

operator

## 5.53.2.6 void AK\_op\_projection\_test ( )

Function for projection operator testing.

## Author

Dino Laktašić

## Returns

No return value

5.53.2.7 char\* AK\_perform\_operatrion ( char \* *op*, struct AK\_operand \* *ab*, struct AK\_operand \* *bb*, int *type* )

Performes arithmetics operation on operand data.

Leon Palačić

## Parameters

<i>ab</i>	first operand
<i>bb</i>	second operand
<i>op</i>	operator

## Returns

result of arithmetics

5.53.2.8 `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

<i>att</i>	- list of atributes on which we make projection
<i>dstTable</i>	table name for projection table

## Returns

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

5.53.2.9 `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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

## Returns

No return value

5.53.2.10 `void removeSubstring ( char * s, const char * toremove )`

Function that removes specified part of string.

Leon Palaić

## Parameters

<i>s</i>	input string
<i>toremove</i>	remove string

## Returns

new sequence of characters

## 5.54 rel/projection.h File Reference

```
#include "expression_check.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
```

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

## Classes

- struct [AK\\_operand](#)

## 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.*
- int [AK\\_determine\\_header\\_type](#) (int a, int b)  
*Determines new header type.*
- char \* [AK\\_create\\_header\\_name](#) (char \*first, char \*operator, char \*second)  
*Creates new header name from passed operand names and operator.*
- char \* [AK\\_perform\\_operatrion](#) (char \*op, struct [AK\\_operand](#) \*a, struct [AK\\_operand](#) \*b, int type)  
*Performes arithmetics operation on operand data.*
- void [removeSubstring](#) (char \*s, const char \*toremove)  
*Function that removes specified part of string.*
- void [AK\\_op\\_projection\\_test](#) ()  
*Function for projection operator testing.*
- char \* [AK\\_get\\_operator](#) (char \*exp)  
*Function that fetches arithmetics operator.*

### 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

<i>old_block</i>	block from which we copy data
<i>dstTable</i>	name of the new table
<i>att</i>	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

<i>old_block_add</i>	address of the block from which we copy headers we need
<i>dstTable</i>	name of the new table
<i>att</i>	list of the attributes which should the projeciton table contain

Returns

No return value

5.54.2.3 char\* AK\_create\_header\_name ( char \* *first*, char \* *second*, char \* *operator* )

Creates new header name from passed operand names and operator.

Leon Palaić

Parameters

<i>first</i>	operand name
<i>second</i>	operand name
<i>operator</i>	

Returns

new sequence of characters

5.54.2.4 int AK\_determine\_header\_type ( int *a*, int *b* )

Determines new header type.

Leon Palaić

## Parameters

<i>a</i>	operand type
<i>b</i>	operand type

## Returns

header type

5.54.2.5 `char* AK_get_operator ( char * exp )`

Function that fetches arithmetics operator.

Leon Palaić

## Parameters

<i>exp</i>	input expression string
------------	-------------------------

## Returns

operator

5.54.2.6 `void AK_op_projection_test ( )`

Function for projection operator testing.

## Author

Dino Laktašić

## Returns

No return value

5.54.2.7 `char* AK_perform_operatrion ( char * op, struct AK_operand * ab, struct AK_operand * bb, int type )`

Performes arithmetics operation on operand data.

Leon Palaić

## Parameters

<i>ab</i>	first operand
<i>bb</i>	second operand
<i>op</i>	operator

## Returns

result of arithmetics

5.54.2.8 `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

<i>att</i>	- list of atributes on which we make projection
<i>dstTable</i>	table name for projection table

## Returns

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

5.54.2.9 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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

## Returns

No return value

5.54.2.10 void removeSubstring ( char \* *s*, const char \* *toremove* )

Function that removes specified part of string.

Leon Palać

## Parameters

<i>s</i>	input string
<i>toremove</i>	remove string

## Returns

new sequence of characters

## 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

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with postfix 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**

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with postfix 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

<i>name</i>	name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	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

<i>name</i>	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

<i>name</i>	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**

<i>name</i>	Name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	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**

<i>name</i>	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**

<i>name</i>	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

<i>old_name</i>	Name of the sequence to be renamed
<i>new_name</i>	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 "../aux/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

<i>name</i>	name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	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

<i>name</i>	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

<i>name</i>	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**

<i>name</i>	Name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	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**

<i>name</i>	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**

<i>name</i>	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

<i>old_name</i>	Name of the sequence to be renamed
<i>new_name</i>	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 between two tables on specified conditions. Names of the attributes 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>tbl1_num_att</i>	number of attributes in the first table
<i>tbl2_num_att</i>	number of attributes in the second table
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>new_table</i>	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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>new_table</i>	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 between two tables on specified conditions. Names of the attributes 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

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>dstTable</i>	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 "../aux/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 between two tables on specified conditions. Names of the attributes 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>tbl1_num_att</i>	number of attributes in the first table
<i>tbl2_num_att</i>	number of attributes in the second table
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>new_table</i>	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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>new_table</i>	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 between two tables on specified conditions. Names of the attributes 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**

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>dstTable</i>	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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 "../aux/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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	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 particular attribute, string constraint should be written 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\\_print\\_constraints](#) (char \*tableName)  
*Function for printing tables.*
- int [AK\\_delete\\_constraint\\_between](#) (char \*tableName, char \*constraintNamePar, char \*attNamePar)  
*Function for deleting specific between constraint.*
- void [Ak\\_constraint\\_between\\_test](#) ()  
*Tests the functionality of implemented between constraint.*

### 5.63.1 Detailed Description

Provides functions for between constraint

### 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\_delete\_constraint\_between ( char \* *tableName*, char \* *constraintNamePar*, char \* *attNamePar* )

Function for deleting specific between constraint.

##### Author

Maja Vračan

##### Parameters

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

##### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

#### 5.63.2.3 int AK\_find\_table\_address ( char \* *\_systemTableName* )

Returns system tables address by name.

##### Author

Mislav Jurinić

##### Parameters

<i>_systemTable-Name</i>	table name
--------------------------	------------

##### Returns

int

#### 5.63.2.4 void AK\_print\_constraints ( char \* *tableName* )

Function for printing tables.

##### Author

Maja Vračan

##### Parameters

<i>tableName</i>	name of table
------------------	---------------

#### 5.63.2.5 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

<i>tableName</i>	table name
<i>newValue</i>	value we want to insert
<i>attNamePar</i>	attribute name

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

5.63.2.6 void **AK\_set\_constraint\_between** ( char \* *tableName*, char \* *constraintName*, char \* *attName*, char \* *startValue*, char \* *endValue* )

Function sets between constraints on particular attribute, string constraint should be written 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

<i>tableName</i>	table name
<i>constraintName</i>	name of constraint
<i>attName</i>	name of attribute
<i>startValue</i>	initial constraint
<i>endValue</i>	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 particular attribute, string constraint should be written 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.*
- int [AK\\_delete\\_constraint\\_between](#) (char \* *tableName*, char \* *attName*[], char \* *constraintName*[])
- 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 constraint

### 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

<i>_systemTableName</i>	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

<i>tableName</i>	table name
<i>newValue</i>	value we want to insert
<i>attNamePar</i>	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 particular attribute, string constraint should be written 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

<i>tableName</i>	table name
<i>constraintName</i>	name of constraint
<i>attName</i>	name of attribute
<i>startValue</i>	initial constraint
<i>endValue</i>	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

<i>table</i>	target table name
<i>attribute</i>	target attribute name
<i>value</i>	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

<i>table_name</i>	target table for "check" constraint evaluation
<i>constraint_name</i>	new "check" constraint name that will be visible in the system table
<i>attribute_name</i>	target attribute for "check" constraint evaluation
<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	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**

<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set
<i>row_data</i>	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 "../..aux/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**

<i>table</i>	target table name
<i>attribute</i>	target attribute name



<i>value</i>	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**

<i>table_name</i>	target table for "check" constraint evaluation
<i>constraint_name</i>	new "check" constraint name that will be visible in the system table
<i>attribute_name</i>	target attribute for "check" constraint evaluation
<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	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**

<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set
<i>row_data</i>	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

<i>char</i>	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**

<i>char</i>	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/nnnull.c File Reference

```
#include "nnnull.h"
Include dependency graph for nnnull.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.*
- int [AK\\_delete\\_constraint\\_not\\_null](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific 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 int [AK\\_delete\\_constraint\\_not\\_null](#) ( char \* *tableName*, char *attName*[], char *constraintName*[] )

Function for deleting specific not null constraint.

#### Author

Maja Vračan

#### Parameters

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

#### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

5.69.2.2 void [AK\\_null\\_test](#) ( )

Function for testing testing NOT NULL constraint.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Returns

No return value

### 5.69.2.3 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

<i>char*</i>	<i>tableName</i> name of table
<i>char*</i>	<i>attName</i> name of attribute
<i>char*</i>	<i>newValue</i> new value

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

### 5.69.2.4 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

<i>char*</i>	<i>tableName</i> name of table
<i>char*</i>	<i>attName</i> name of attribute
<i>char*</i>	<i>constraintName</i> 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 "../aux/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.*
- int [AK\\_delete\\_constraint\\_not\\_null](#) (char \**tableName*, char *attName*[], char *constraintName*[])  
*Function for deleting specific 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 `int AK_delete_constraint_not_null ( char * tableName, char attName[], char constraintName[] )`

Function for deleting specific not null constraint.

##### Author

Maja Vračan

##### Parameters

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

##### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

#### 5.70.2.2 `void AK_null_test ( )`

Function for testing testing NOT NULL constraint.

##### Author

Saša Vukšić, updated by Nenad Makar

##### Returns

No return value

#### 5.70.2.3 `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

<i>char*</i>	<i>tableName</i> name of table
<i>char*</i>	<i>attName</i> name of attribute
<i>char*</i>	<i>newValue</i> new value

##### Returns

EXIT\_ERROR or EXIT\_SUCCESS

## 5.70.2.4 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 applicable 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

<i>name</i>	of the child table
<i>array</i>	of child table attribute names (foreign key attributes)
<i>name</i>	of the parent table
<i>array</i>	of parent table attribute names (primary key attributes)
<i>number</i>	of attributes in foreign key
<i>name</i>	of the constraint
<i>type</i>	of the constraint, constants defined in ' <a href="#">reference.h</a> '

### 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

<i>name</i>	of the table with reference (with foreign key)
<i>name</i>	of the reference constraint

### Returns

[AK\\_ref\\_item](#) object with all necessary 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

<i>child</i>	table name
<i>attribute</i>	name (foreign key attribute)
<i>value</i>	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

<i>list</i>	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* )

Function that quickly checks if there are any referential constraints that should be applied on a given list of changes.

## Author

Dejan Frankovic

## Parameters

<i>list</i>	of elements for update
<i>is</i>	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 applicable to the operation of updating or deleting a row in a table.

## Author

Dejan Franković

## Parameters

<i>list</i>	of elements for update
<i>is</i>	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

<i>list</i>	of elements for update
<i>is</i>	action UPDATE or DELETE ?

## Returns

EXIT\_SUCCESS

## 5.72 sql/cs/reference.h File Reference

```
#include "../dm/dbman.h"
#include "../file/table.h"
#include "../aux/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)`  
*Function 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 applicable 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 elements 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 creating new table, index, temporary table, etc. call this function.

### 5.72.1 Detailed Description

d 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

<i>name</i>	of the child table
<i>array</i>	of child table attribute names (foreign key attributes)
<i>name</i>	of the parent table
<i>array</i>	of parent table attribute names (primary key attributes)
<i>number</i>	of attributes in foreign key
<i>name</i>	of the constraint
<i>type</i>	of the constraint, constants defined in ' <a href="#">reference.h</a> '

#### 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

<i>row_root</i>	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

<i>name</i>	of the table with reference (with foreign key)
<i>name</i>	of the reference constraint

## Returns

[AK\\_ref\\_item](#) object with all necessary 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 creating new table, index, temporary table, etc. call this function.

## Author

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

## Parameters

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	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

<i>row_root</i>	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

<i>child</i>	table name
<i>attribute</i>	name (foreign key attribute)
<i>value</i>	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

<i>list</i>	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

<i>list</i>	of elements for update
<i>is</i>	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

<i>list</i>	of elements for update
<i>is</i>	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

<i>list</i>	of elements for update
<i>is</i>	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

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name



<i>*expr</i>	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**

<i>row_root</i>	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.*
- int [AK\\_delete\\_constraint\\_unique](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific 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\_delete\_constraint\_unique ( char \* tableName, char attName[], char constraintName[] )

Function for deleting specific unique constraint.

**Author**

Maja Vračan

## Parameters

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

## Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

### 5.73.2.2 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

<i>char*</i>	<i>tableName</i> name of table
<i>char</i>	<i>attName</i> [] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)
<i>char</i>	<i>newValue</i> [] new value(s), if you want to check combination of values of more attributes separate 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 <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with sprintf in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in <i>newValue</i> [] use strcat(destination, source) and put constant SEPARATOR between them (see test) if <i>newValue</i> [] should contain NULL sign pass it as " " (space)

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

### 5.73.2.3 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

<i>char*</i>	<i>tableName</i> name of table
<i>char</i>	<i>attName</i> [] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes separate their names with constant SEPARATOR (see test)
<i>char</i>	<i>constraintName</i> [] name of constraint

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

## 5.73.2.4 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 "../aux/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.*
- int [AK\\_delete\\_constraint\\_unique](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific 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\_delete\_constraint\_unique ( char \* tableName, char attName[], char constraintName[] )

Function for deleting specific unique constraint.

## Author

Maja Vračan

## Parameters

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

## Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

#### 5.74.2.2 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

<i>char*</i>	<i>tableName</i> name of table
<i>char</i>	<i>attName</i> [] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)
<i>char</i>	<i>newValue</i> [] new value(s), if you want to check combination of values of more attributes separate 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 <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with sprintf in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in <i>newValue</i> [] use strcat(destination, source) and put constant SEPARATOR between them (see test) if <i>newValue</i> [] should contain NULL sign pass it as " " (space)

##### Returns

EXIT\_ERROR or EXIT\_SUCCESS

#### 5.74.2.3 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

<i>char*</i>	<i>tableName</i> name of table
<i>char</i>	<i>attName</i> [] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes separate their names with constant SEPARATOR (see test)
<i>char</i>	<i>constraintName</i> [] name of constraint

##### Returns

EXIT\_ERROR or EXIT\_SUCCESS

#### 5.74.2.4 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

- int [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 int [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, Maja Vračan

##### Parameters

<i>type</i>	drop type
<a href="#">drop_arguments</a>	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**

<i>tblName</i>	name of table or index
<i>sys_table</i>	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, Maja Vračan

**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**

<i>tblName</i>	name of table, index view, function, trigger, sequence, user, group or constraint
<i>sys_table</i>	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

- int [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 int [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, Maja Vračan

#### Parameters

<i>type</i>	drop type
<a href="#">drop_arguments</a>	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, Maja Vračan

### 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

<i>tblName</i>	name of table, index view, function, trigger, sequence, user, group or constraint
<i>sys_table</i>	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

<i>*function_id</i>	id of the function
<i>*arguments_list</i>	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

<i>function_id</i>	id of the function
<i>args</i>	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

<i>*name</i>	name of the function
<i>*return_type</i>	data type returned from a function - values from 0 to 13 - defined in constants.h
<i>*arguments_list</i>	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

<i>*function_id</i>	id of the function to which the argument belongs
<i>*arg_number</i>	number of the argument
<i>*arg_type</i>	data type of the argument
<i>*argname</i>	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

<i>obj_id</i>	<i>obj_id</i> 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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of function arguments
<i>*new_return_type</i>	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

<i>*name</i>	name of the function
<i>*arguments_list</i>	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

<i>obj_id</i>	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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of arguments to be modified
<i>*new_name</i>	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**

<i>*function</i>	name of the function
<i>*arguments_list</i>	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 "../aux/mempro.h"
#include "../aux/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

<i>*function_id</i>	id of the function
<i>*arguments_list</i>	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

<i>function_id</i>	id of the function
<i>args</i>	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

<i>*name</i>	name of the function
<i>*return_type</i>	data type returned from a function - values from 0 to 13 - defined in constants.h
<i>*arguments_list</i>	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

<i>*function_id</i>	id of the function to which the argument belongs
<i>*arg_number</i>	number of the argument
<i>*arg_type</i>	data type of the argument
<i>*argname</i>	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

<i>obj_id</i>	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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of function arguments
<i>*new_return_type</i>	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

<i>*name</i>	name of the function
<i>*arguments_list</i>	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

<i>obj_id</i>	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**

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of arguments to be modified
<i>*new_name</i>	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**

<i>*function</i>	name of the function
<i>*arguments_list</i>	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**

<i>*user</i>	username of user which will be put in group
<i>*group</i>	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**

<i>group</i>	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**

<i>*user</i>	username for which we want check privileges
<i>*table</i>	name of table for which we want to check whether user has right on
<i>*privilege</i>	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

<i>user</i>	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

<i>*groupname</i>	name of group to which we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	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

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	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

<i>*name</i>	name of group to be added
<i>set_id</i>	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

<i>*name</i>	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

<i>name</i>	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

---

<i>old_name</i>	Name of the group to be renamed
<i>new_name</i>	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**

<i>group</i>	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**

<i>user</i>	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

<i>groupname</i>	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

<i>username</i>	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

<i>*grounamep</i>	name of group which user belongs to
<i>*table</i>	name of table on which privilege will be granted to group
<i>*right</i>	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

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be revoked from user
<i>*right</i>	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

<i>*username</i>	username of user to be added
<i>password</i>	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

<i>*username</i>	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 Barač

## Parameters

<i>name</i>	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

<i>old_name</i>	Name of the user to be renamed
<i>new_name</i>	New name of the user
<i>password</i>	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

<i>srcTable</i>	- original table that is used for selection
<i>destTable</i>	- table that contains the result
<i>condition</i>	- 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

<i>*name</i>	name of the trigger
<i>*event</i>	event that calls the trigger - this should perhaps be an integer with defined constants...
<i>*condition</i>	AK_list list of conditions in postfix
<i>*table</i>	name of the table trigger is hooked on
<i>*function</i>	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

<i>*name</i>	name of the trigger (or NULL if using obj_id)
<i>*event</i>	event of the trigger (or NULL if it isn't changing)
<i>*condition</i>	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)

<i>*table</i>	name of the connected table (or NULL id using obj_id)
<i>*function</i>	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**

<i>trigger</i>	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**

<i>*name</i>	name of the trigger
<i>*table</i>	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**

<i>*name</i>	name of the trigger
<i>*table</i>	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

<i>obj_id</i>	<i>obj_id</i> 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

<i>old_name</i>	Name of the trigger to be renamed
<i>new_name</i>	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

<i>trigger</i>	<i>obj_id</i> of the trigger in question
* <i>condition</i>	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**

<i>*name</i>	name of the trigger
<i>*event</i>	event that calls the trigger - this should perhaps be an integer with defined constants...
<i>*condition</i>	AK_list list of conditions in postfix
<i>*table</i>	name of the table trigger is hooked on
<i>*function</i>	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**

<i>*name</i>	name of the trigger (or NULL if using obj_id)
<i>*event</i>	event of the trigger (or NULL if it isn't changing)
<i>*condition</i>	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
<i>*table</i>	name of the connected table (or NULL id using obj_id)
<i>*function</i>	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**

<i>trigger</i>	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

<i>*name</i>	name of the trigger
<i>*table</i>	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

<i>*name</i>	name of the trigger
<i>*table</i>	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

<i>obj_id</i>	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

<i>old_name</i>	Name of the trigger to be renamed
<i>new_name</i>	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**

<i>trigger</i>	obj_id of the trigger in question
* <i>condition</i>	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 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

<i>name</i>	name of the view
-------------	------------------

##### Returns

View's id or EXIT\_ERROR

#### 5.83.2.3 char\* AK\_get\_view\_query ( char \* name )

Returns a query by its name.

##### Author

Danko Sačer

## Parameters

<i>name</i>	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

<i>name</i>	name og the view
<i>query</i>	query of the view
<i>rel_exp</i>	relation expression of the view
<i>set_id</i>	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

<i>name</i>	of the query
<i>query</i>	new query of the view
<i>rel_exp</i>	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

<i>name</i>	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

<i>obj_id</i>	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

<i>name</i>	name of the view
<i>new_name</i>	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\_transaction 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 **endTransactionTestLockMutex** = 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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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

<i>blockAddress</i>	integer representation of memory address.
<i>type</i>	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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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

<i>transaction_data</i>	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

<i>blockAddress</i>	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

<i>blockAddress</i>	integer representation of memory address.
<i>id</i>	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

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	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

<i>data</i>	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

<i>addressList</i>	pointer to the linked list where the addresses are stored.
<i>tblName</i>	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

<i>noticeType</i>	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

<i>lockHolder</i>	pointer to the hash list entry that is entitled to the specific memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.
<i>lock</i>	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

<i>blockMemory-Address</i>	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

<i>observer</i>	Observer type
<i>observable</i>	Observable type
<i>type</i>	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

<i>transaction_ - thread</i>	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

<i>adreses</i>	linked list of memory addresses locked by the transaction.
<i>transactionId</i>	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

<i>transaction_ - thread</i>	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

<i>blockAddress</i>	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

<i>blockAddress</i>	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

<i>memoryAddress</i>	integer representation of memory address.
<i>id</i>	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

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	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

<i>observable_transaction</i>	Observable type instance
<i>observer</i>	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\_transaction type.

## Author

Ivan Pusic

## Parameters

<i>observable_transaction</i>	Observable type instance
<i>observer</i>	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

<a href="#">observer_lock</a>	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\_transaction 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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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

<i>blockAddress</i>	integer representation of memory address.
<i>type</i>	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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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**

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	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**

<i>transaction_data</i>	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**

<i>blockAddress</i>	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**

<i>blockAddress</i>	integer representation of memory address.
<i>id</i>	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**

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	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**

<i>data</i>	transmitted to the thread from the main thread
-------------	--

#### 5.85.3.11 int AK\_get\_memory\_blocks ( char \* tbName, AK\_memoryAddresses\_link addressList )

Method that appends all addresses affected by the transaction.

**Author**

Frane Jakelić

**Parameters**

<i>addressList</i>	pointer to the linked list where the addresses are stored.
<i>tblName</i>	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**

<i>noticeType</i>	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

<i>lockHolder</i>	pointer to the hash list entry that is entitled to the specific memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.
<i>lock</i>	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

<i>blockMemory-Address</i>	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**

<i>observer</i>	Observer type
<i>observable</i>	Observable type
<i>type</i>	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**

<i>transaction_ - thread</i>	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**

<i>adresses</i>	linked list of memory addresses locked by the transaction.
<i>transactionId</i>	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**

<i>transaction_ - thread</i>	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**

<i>blockAddress</i>	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**

<i>blockAddress</i>	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**

<i>memoryAddress</i>	integer representation of memory address.
<i>id</i>	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

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	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

<i>observable_transaction</i>	Observable type instance
<i>observer</i>	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\_transaction type.

Author

Ivan Pusic

Parameters

<i>observable_transaction</i>	Observable type instance
<i>observer</i>	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

<a href="#"><i>observer_lock</i></a>	Observer type instance
--------------------------------------	------------------------

# Index

- [\\_file\\_metadata](#), 9
- [AK\\_GUID](#)
  - [blobs.c](#), 61
  - [blobs.h](#), 64
- [AK\\_acquire\\_lock](#)
  - [transaction.c](#), 287
  - [transaction.h](#), 298
- [AK\\_add\\_hash\\_entry\\_list](#)
  - [transaction.c](#), 287
  - [transaction.h](#), 300
- [AK\\_add\\_lock](#)
  - [transaction.c](#), 288
  - [transaction.h](#), 300
- [AK\\_add\\_reference](#)
  - [reference.c](#), 242
  - [reference.h](#), 246
- [AK\\_add\\_to\\_bitmap\\_index](#)
  - [bitmap.c](#), 85
  - [bitmap.h](#), 89
- [AK\\_add\\_to\\_redolog](#)
  - [redo\\_log.c](#), 182
- [AK\\_add\\_user\\_to\\_group](#)
  - [privileges.c](#), 267
- [AK\\_agg\\_input](#), 9
- [AK\\_agg\\_input\\_add](#)
  - [aggregation.c](#), 184
  - [aggregation.h](#), 188
- [AK\\_agg\\_input\\_add\\_to\\_beginning](#)
  - [aggregation.c](#), 184
  - [aggregation.h](#), 188
- [AK\\_agg\\_input\\_fix](#)
  - [aggregation.c](#), 184
  - [aggregation.h](#), 189
- [AK\\_agg\\_input\\_init](#)
  - [aggregation.c](#), 185
  - [aggregation.h](#), 189
- [AK\\_agg\\_value](#), 10
- [AK\\_aggregation](#)
  - [aggregation.c](#), 185
  - [aggregation.h](#), 189
- [AK\\_all\\_transactions\\_finished](#)
  - [transaction.c](#), 288
  - [transaction.h](#), 300
- [AK\\_allocate\\_block\\_activity\\_modes](#)
  - [dbman.c](#), 33
- [AK\\_allocate\\_blocks](#)
  - [dbman.c](#), 33
  - [dbman.h](#), 48
- [AK\\_allocation\\_set\\_mode](#)
  - [dbman.h](#), 48
- [AK\\_allocationbit](#)
  - [dbman.h](#), 59
- [AK\\_allocationtable\\_dump](#)
  - [dbman.c](#), 33
  - [dbman.h](#), 48
- [AK\\_archive\\_log](#)
  - [archive\\_log.h](#), 179
- [AK\\_block](#), 10
- [AK\\_block\\_activity](#), 11
- [AK\\_block\\_sort](#)
  - [filesort.h](#), 81
- [AK\\_blocktable](#), 12
- [AK\\_blocktable\\_dump](#)
  - [dbman.c](#), 34
  - [dbman.h](#), 48
- [AK\\_blocktable\\_flush](#)
  - [dbman.c](#), 34
  - [dbman.h](#), 48
- [AK\\_blocktable\\_get](#)
  - [dbman.c](#), 34
  - [dbman.h](#), 49
- [AK\\_btree\\_create](#)
  - [btree.c](#), 94
  - [btree.h](#), 95
- [AK\\_btree\\_search\\_delete](#)
  - [btree.c](#), 94
  - [btree.h](#), 95
- [AK\\_cache\\_AK\\_malloc](#)
  - [memoman.c](#), 141
  - [memoman.h](#), 147
- [AK\\_cache\\_block](#)
  - [memoman.c](#), 141
  - [memoman.h](#), 148
- [AK\\_cache\\_result](#)
  - [memoman.c](#), 141
  - [memoman.h](#), 148
- [AK\\_change\\_hash\\_info](#)
  - [hash.c](#), 96
  - [hash.h](#), 101
- [AK\\_check\\_arithmetic\\_statement](#)
  - [expression\\_check.c](#), 193
  - [expression\\_check.h](#), 195
- [AK\\_check\\_attributes](#)
  - [redo\\_log.c](#), 183
- [AK\\_check\\_constraint](#)
  - [check\\_constraint.c](#), 232
  - [check\\_constraint.h](#), 234
- [AK\\_check\\_constraint\\_test](#)

- check\_constraint.c, [233](#)
- check\_constraint.h, [235](#)
- AK\_check\_constraints
  - theta\_join.c, [222](#)
  - theta\_join.h, [224](#)
- AK\_check\_folder\_blobs
  - blobs.c, [60](#)
  - blobs.h, [63](#)
- AK\_check\_function\_arguments
  - function.c, [259](#)
  - function.h, [263](#)
- AK\_check\_function\_arguments\_type
  - function.c, [259](#)
  - function.h, [263](#)
- AK\_check\_group\_privilege
  - privileges.c, [268](#)
- AK\_check\_if\_row\_satisfies\_expression
  - expression\_check.c, [193](#)
  - expression\_check.h, [195](#)
- AK\_check\_privilege
  - privileges.c, [268](#)
- AK\_check\_tables\_scheme
  - table.c, [118](#)
  - table.h, [127](#)
- AK\_check\_user\_privilege
  - privileges.c, [268](#)
- AK\_command\_recovery\_struct, [12](#)
- AK\_command\_struct, [12](#)
- AK\_compare
  - rel\_eq\_assoc.c, [157](#)
  - rel\_eq\_assoc.h, [158](#)
- AK\_concat
  - blobs.c, [60](#)
  - blobs.h, [63](#)
- AK\_constraint\_names\_test
  - constraint\_names.c, [236](#)
  - constraint\_names.h, [237](#)
- AK\_copy\_block\_projection
  - projection.c, [206](#)
  - projection.h, [211](#)
- AK\_copy\_blocks\_join
  - nat\_join.c, [199](#)
  - nat\_join.h, [201](#)
- AK\_copy\_header
  - dbman.c, [34](#)
  - dbman.h, [49](#)
- AK\_create\_Index\_Table
  - bitmap.c, [86](#)
  - bitmap.h, [90](#)
- AK\_create\_block\_header
  - projection.c, [206](#)
  - projection.h, [211](#)
- AK\_create\_hash\_index
  - hash.c, [97](#)
  - hash.h, [101](#)
- AK\_create\_header
  - dbman.c, [35](#)
  - dbman.h, [49](#)
- AK\_create\_header\_name
  - projection.c, [206](#)
  - projection.h, [211](#)
- AK\_create\_join\_block\_header
  - nat\_join.c, [199](#)
  - nat\_join.h, [202](#)
- AK\_create\_lock
  - transaction.c, [288](#)
  - transaction.h, [300](#)
- AK\_create\_new\_transaction\_thread
  - transaction.c, [288](#)
  - transaction.h, [301](#)
- AK\_create\_table\_struct, [13](#)
- AK\_create\_test\_tables
  - test.c, [135](#)
  - test.h, [137](#)
- AK\_create\_theta\_join\_header
  - theta\_join.c, [223](#)
  - theta\_join.h, [225](#)
- AK\_db\_cache, [13](#)
- AK\_deallocate\_search\_result
  - filesearch.c, [77](#)
  - filesearch.h, [79](#)
- AK\_delete\_bitmap\_index
  - bitmap.c, [86](#)
  - bitmap.h, [91](#)
- AK\_delete\_block
  - dbman.c, [35](#)
  - dbman.h, [50](#)
- AK\_delete\_constraint\_between
  - between.c, [228](#)
- AK\_delete\_constraint\_not\_null
  - nnull.c, [238](#)
  - nnull.h, [240](#)
- AK\_delete\_constraint\_unique
  - unique.c, [251](#)
  - unique.h, [253](#)
- AK\_delete\_extent
  - dbman.c, [35](#)
  - dbman.h, [50](#)
- AK\_delete\_hash\_entry\_list
  - transaction.c, [289](#)
  - transaction.h, [301](#)
- AK\_delete\_in\_hash\_index
  - hash.c, [97](#)
  - hash.h, [102](#)
- AK\_delete\_lock\_entry\_list
  - transaction.c, [289](#)
  - transaction.h, [301](#)
- AK\_delete\_segment
  - dbman.c, [36](#)
  - dbman.h, [50](#)
- AK\_determine\_header\_type
  - projection.c, [208](#)
  - projection.h, [211](#)
- AK\_difference
  - difference.c, [191](#)
  - difference.h, [192](#)

- AK\_drop
  - drop.c, [255](#)
  - drop.h, [257](#)
- AK\_drop\_help\_function
  - drop.c, [255](#)
- AK\_drop\_test
  - drop.c, [256](#)
  - drop.h, [257](#)
- AK\_elem\_hash\_value
  - hash.c, [97](#)
  - hash.h, [102](#)
- AK\_empty\_archive\_log
  - archive\_log.h, [179](#)
- AK\_execute\_commands
  - transaction.c, [289](#)
  - transaction.h, [302](#)
- AK\_execute\_rel\_eq
  - query\_optimization.c, [153](#)
  - query\_optimization.h, [155](#)
- AK\_execute\_transaction
  - transaction.c, [290](#)
  - transaction.h, [302](#)
- AK\_find\_AK\_free\_space
  - memoman.c, [141](#)
  - memoman.h, [148](#)
- AK\_find\_available\_result\_block
  - memoman.c, [142](#)
  - memoman.h, [148](#)
- AK\_find\_delete\_in\_hash\_index
  - hash.c, [98](#)
  - hash.h, [102](#)
- AK\_find\_in\_hash\_index
  - hash.c, [98](#)
  - hash.h, [103](#)
- AK\_find\_table\_address
  - between.c, [229](#)
  - between.h, [231](#)
- AK\_flush\_cache
  - memoman.c, [142](#)
  - memoman.h, [149](#)
- AK\_folder\_exists
  - blobs.c, [60](#)
  - blobs.h, [64](#)
- AK\_function\_add
  - function.c, [259](#)
  - function.h, [263](#)
- AK\_function\_arguments\_add
  - function.c, [259](#)
  - function.h, [264](#)
- AK\_function\_arguments\_remove\_by\_obj\_id
  - function.c, [260](#)
  - function.h, [264](#)
- AK\_function\_change\_return\_type
  - function.c, [260](#)
  - function.h, [264](#)
- AK\_function\_remove\_by\_name
  - function.c, [260](#)
  - function.h, [265](#)
- AK\_function\_remove\_by\_obj\_id
  - function.c, [261](#)
  - function.h, [265](#)
- AK\_function\_rename
  - function.c, [261](#)
  - function.h, [265](#)
- AK\_function\_test
  - function.c, [261](#)
  - function.h, [266](#)
- AK\_generate\_result\_id
  - memoman.c, [142](#)
  - memoman.h, [149](#)
- AK\_get\_Attribute
  - bitmap.c, [86](#)
  - bitmap.h, [91](#)
- AK\_get\_allocation\_set
  - dbman.c, [36](#)
  - dbman.h, [51](#)
- AK\_get\_attr\_index
  - table.c, [118](#)
  - table.h, [127](#)
- AK\_get\_attr\_name
  - table.c, [118](#)
  - table.h, [127](#)
- AK\_get\_block
  - memoman.c, [142](#)
  - memoman.h, [149](#)
- AK\_get\_column
  - table.c, [119](#)
  - table.h, [128](#)
- AK\_get\_extent
  - dbman.c, [36](#)
  - dbman.h, [51](#)
- AK\_get\_function\_obj\_id
  - function.c, [262](#)
  - function.h, [266](#)
- AK\_get\_hash\_info
  - hash.c, [98](#)
  - hash.h, [103](#)
- AK\_get\_header
  - table.c, [119](#)
  - table.h, [128](#)
- AK\_get\_id
  - id.c, [82](#)
  - id.h, [83](#)
- AK\_get\_index\_addresses
  - memoman.c, [143](#)
  - memoman.h, [149](#)
- AK\_get\_index\_header
  - index.c, [107](#)
- AK\_get\_index\_num\_records
  - index.c, [107](#)
  - index.h, [113](#)
- AK\_get\_index\_segment\_addresses
  - memoman.c, [143](#)
  - memoman.h, [150](#)
- AK\_get\_index\_tuple
  - index.c, [107](#)

- index.h, 113
- AK\_get\_memory\_blocks
  - transaction.c, 290
  - transaction.h, 302
- AK\_get\_num\_records
  - table.c, 119
  - table.h, 128
- AK\_get\_operator
  - projection.c, 208
  - projection.h, 212
- AK\_get\_reference
  - reference.c, 242
  - reference.h, 247
- AK\_get\_rel\_exp
  - view.c, 283
- AK\_get\_row
  - table.c, 120
  - table.h, 129
- AK\_get\_segment\_addresses
  - memoman.c, 143
  - memoman.h, 150
- AK\_get\_table\_addresses
  - memoman.c, 144
  - memoman.h, 150
- AK\_get\_table\_attribute\_types
  - test.c, 135
  - test.h, 138
- AK\_get\_table\_id
  - id.c, 82
- AK\_get\_table\_obj\_id
  - table.c, 120
  - table.h, 129
- AK\_get\_timestamp
  - archive\_log.h, 179
- AK\_get\_tuple
  - table.c, 120
  - table.h, 129
- AK\_get\_view\_obj\_id
  - view.c, 283
- AK\_get\_view\_query
  - view.c, 283
- AK\_grant\_privilege\_group
  - privileges.c, 269
- AK\_grant\_privilege\_user
  - privileges.c, 269
- AK\_group\_add
  - privileges.c, 269
- AK\_group\_get\_id
  - privileges.c, 270
- AK\_group\_remove\_by\_name
  - privileges.c, 270
- AK\_group\_rename
  - privileges.c, 270
- AK\_handle\_observable\_transaction\_action
  - transaction.c, 290
  - transaction.h, 303
- AK\_header, 13
- AK\_header\_size
  - aggregation.c, 186
  - aggregation.h, 190
- AK\_if\_exist
  - drop.c, 256
  - drop.h, 257
- AK\_increase\_extent
  - dbman.c, 37
  - dbman.h, 51
- AK\_index\_table\_exist
  - index.c, 109
  - index.h, 115
- AK\_index\_test
  - index.c, 109
  - index.h, 115
- AK\_init\_allocation\_table
  - dbman.c, 37
  - dbman.h, 52
- AK\_init\_block
  - dbman.c, 38
  - dbman.h, 52
- AK\_init\_db\_file
  - dbman.c, 38
  - dbman.h, 52
- AK\_init\_disk\_manager
  - dbman.c, 38
  - dbman.h, 53
- AK\_init\_new\_extent
  - memoman.c, 144
  - memoman.h, 151
- AK\_init\_observable\_transaction
  - transaction.c, 290
  - transaction.h, 303
- AK\_init\_observer\_lock
  - transaction.c, 291
  - transaction.h, 303
- AK\_init\_system\_catalog
  - dbman.c, 38
  - dbman.h, 53
- AK\_init\_system\_tables\_catalog
  - dbman.c, 39
  - dbman.h, 53
- AK\_initialize\_new\_index\_segment
  - files.c, 74
  - files.h, 76
- AK\_initialize\_new\_segment
  - files.c, 75
  - files.h, 76
  - reference.h, 247
- AK\_insert\_entry
  - dbman.c, 40
  - dbman.h, 54
- AK\_insert\_in\_hash\_index
  - hash.c, 99
  - hash.h, 104
- AK\_intersect
  - intersect.c, 197
  - intersect.h, 198
- AK\_isLock\_waiting

- transaction.c, [291](#)
- transaction.h, [303](#)
- AK\_join
  - nat\_join.c, [200](#)
  - nat\_join.h, [202](#)
- AK\_lo\_export
  - blobs.c, [61](#)
  - blobs.h, [64](#)
- AK\_lo\_import
  - blobs.c, [61](#)
  - blobs.h, [64](#)
- AK\_lo\_test
  - blobs.c, [61](#)
  - blobs.h, [64](#)
- AK\_lo\_unlink
  - blobs.c, [61](#)
  - blobs.h, [65](#)
- AK\_lock\_released
  - transaction.c, [291](#)
  - transaction.h, [304](#)
- AK\_mem\_block, [14](#)
- AK\_mem\_block\_modify
  - memoman.c, [144](#)
  - memoman.h, [151](#)
- AK\_memoman\_init
  - memoman.c, [145](#)
  - memoman.h, [151](#)
- AK\_memory\_block\_hash
  - transaction.c, [291](#)
  - transaction.h, [304](#)
- AK\_memset\_int
  - dbman.c, [40](#)
  - dbman.h, [55](#)
- AK\_merge\_block\_join
  - nat\_join.c, [200](#)
  - nat\_join.h, [202](#)
- AK\_mkdir
  - blobs.c, [62](#)
  - blobs.h, [65](#)
- AK\_new\_extent
  - dbman.c, [41](#)
  - dbman.h, [55](#)
- AK\_new\_segment
  - dbman.c, [41](#)
  - dbman.h, [56](#)
- AK\_null\_test
  - nnull.c, [238](#)
  - nnull.h, [240](#)
- AK\_num\_attr
  - table.c, [121](#)
  - table.h, [130](#)
- AK\_num\_index\_attr
  - index.c, [110](#)
  - index.h, [116](#)
- AK\_on\_all\_transactions\_end
  - transaction.c, [292](#)
  - transaction.h, [304](#)
- AK\_on\_lock\_release
  - transaction.c, [292](#)
  - transaction.h, [304](#)
- AK\_on\_observable\_notify
  - transaction.c, [292](#)
  - transaction.h, [304](#)
- AK\_on\_transaction\_end
  - transaction.c, [292](#)
  - transaction.h, [305](#)
- AK\_op\_join\_test
  - nat\_join.c, [200](#)
  - nat\_join.h, [203](#)
- AK\_op\_product\_test
  - product.c, [203](#)
  - product.h, [204](#)
- AK\_op\_projection\_test
  - projection.c, [208](#)
  - projection.h, [212](#)
- AK\_op\_rename\_test
  - table.c, [121](#)
  - table.h, [130](#)
- AK\_op\_selection\_test
  - selection.c, [214](#)
  - selection.h, [215](#)
- AK\_op\_selection\_test2
  - selection.c, [214](#)
  - selection.h, [215](#)
- AK\_op\_selection\_test\_redolog
  - selection.c, [214](#)
  - selection.h, [215](#)
- AK\_op\_theta\_join\_test
  - theta\_join.c, [223](#)
  - theta\_join.h, [225](#)
- AK\_op\_union\_test
  - union.c, [226](#)
  - union.h, [227](#)
- AK\_operand, [15](#)
- AK\_perform\_operatrion
  - projection.c, [208](#)
  - projection.h, [212](#)
- AK\_print\_block
  - dbman.c, [42](#)
  - dbman.h, [56](#)
- AK\_print\_constraints
  - between.c, [229](#)
- AK\_print\_index\_table
  - index.c, [110](#)
  - index.h, [116](#)
- AK\_print\_optimized\_query
  - query\_optimization.c, [153](#)
  - query\_optimization.h, [155](#)
- AK\_print\_rel\_eq\_assoc
  - rel\_eq\_assoc.c, [157](#)
  - rel\_eq\_assoc.h, [159](#)
- AK\_print\_rel\_eq\_comut
  - rel\_eq\_comut.c, [160](#)
  - rel\_eq\_comut.h, [162](#)
- AK\_print\_rel\_eq\_projection
  - rel\_eq\_projection.c, [163](#)

- rel\_eq\_projection.h, 167
- AK\_print\_rel\_eq\_selection
  - rel\_eq\_selection.c, 171
  - rel\_eq\_selection.h, 175
- AK\_print\_row
  - table.c, 121
  - table.h, 130
- AK\_print\_row Spacer
  - table.c, 122
  - table.h, 131
- AK\_print\_row Spacer to file
  - table.c, 122
  - table.h, 131
- AK\_print\_row to file
  - table.c, 122
  - table.h, 131
- AK\_print\_table
  - table.c, 123
  - table.h, 132
- AK\_print\_table to file
  - table.c, 123
  - table.h, 132
- AK\_printout\_redolog
  - redo\_log.c, 183
- AK\_privileges\_test
  - privileges.c, 270
- AK\_product
  - product.c, 203
  - product.h, 205
- AK\_projection
  - projection.c, 209
  - projection.h, 212
- AK\_query\_mem, 15
- AK\_query\_mem\_AK\_malloc
  - memoman.c, 145
  - memoman.h, 151
- AK\_query\_mem\_dict, 15
- AK\_query\_mem\_lib, 16
- AK\_query\_mem\_result, 16
- AK\_query\_optimization
  - query\_optimization.c, 153
  - query\_optimization.h, 155
- AK\_query\_optimization\_test
  - query\_optimization.c, 154
  - query\_optimization.h, 156
- AK\_read\_block
  - dbman.c, 42
  - dbman.h, 56
- AK\_read\_block\_for\_testing
  - dbman.c, 42
  - dbman.h, 57
- AK\_read\_constraint\_between
  - between.c, 229
  - between.h, 231
- AK\_read\_constraint\_not\_null
  - nnull.c, 238
  - nnull.h, 240
- AK\_read\_constraint\_unique
  - unique.c, 251
  - unique.h, 253
- AK\_recover\_archive\_log
  - recovery.c, 180
- AK\_recover\_operation
  - recovery.c, 181
- AK\_recovery\_insert\_row
  - recovery.c, 181
- AK\_recovery\_test
  - recovery.c, 181
- AK\_recovery\_tokenize
  - recovery.c, 181
- AK\_redo\_log, 17
- AK\_redo\_log\_AK\_malloc
  - memoman.c, 145
  - memoman.h, 152
- AK\_ref\_item, 17
- AK\_reference\_check\_attribute
  - reference.c, 242
  - reference.h, 248
- AK\_reference\_check\_entry
  - reference.c, 243
  - reference.h, 249
- AK\_reference\_check\_if\_update\_needed
  - reference.c, 243
  - reference.h, 249
- AK\_reference\_check\_restriction
  - reference.c, 243
  - reference.h, 249
- AK\_reference\_test
  - reference.c, 244
  - reference.h, 250
- AK\_reference\_update
  - reference.c, 244
  - reference.h, 250
- AK\_refresh\_cache
  - memoman.c, 145
  - memoman.h, 152
- AK\_register\_system\_tables
  - dbman.c, 42
  - dbman.h, 57
- AK\_rel\_eq\_assoc
  - rel\_eq\_assoc.c, 157
  - rel\_eq\_assoc.h, 159
- AK\_rel\_eq\_assoc\_test
  - rel\_eq\_assoc.c, 157
  - rel\_eq\_assoc.h, 159
- AK\_rel\_eq\_can\_commute
  - rel\_eq\_projection.c, 164
  - rel\_eq\_projection.h, 168
- AK\_rel\_eq\_collect\_cond\_attributes
  - rel\_eq\_projection.c, 164
  - rel\_eq\_projection.h, 168
- AK\_rel\_eq\_commute\_with\_theta\_join
  - rel\_eq\_comut.c, 160
  - rel\_eq\_comut.h, 162
- AK\_rel\_eq\_comut
  - rel\_eq\_comut.c, 160



- rel\_eq\_comut.h, 162
- AK\_rel\_eq\_comut\_test
  - rel\_eq\_comut.c, 161
  - rel\_eq\_comut.h, 162
- AK\_rel\_eq\_cond\_attributes
  - rel\_eq\_selection.c, 172
  - rel\_eq\_selection.h, 176
- AK\_rel\_eq\_get\_attributes\_char
  - rel\_eq\_selection.c, 172
  - rel\_eq\_selection.h, 176
- AK\_rel\_eq\_get\_attributes
  - rel\_eq\_projection.c, 164
  - rel\_eq\_projection.h, 168
- AK\_rel\_eq\_is\_attr\_subset
  - rel\_eq\_selection.c, 172
  - rel\_eq\_selection.h, 176
- AK\_rel\_eq\_is\_subset
  - rel\_eq\_projection.c, 165
  - rel\_eq\_projection.h, 169
- AK\_rel\_eq\_projection
  - rel\_eq\_projection.c, 165
  - rel\_eq\_projection.h, 169
- AK\_rel\_eq\_projection\_attributes
  - rel\_eq\_projection.c, 166
  - rel\_eq\_projection.h, 170
- AK\_rel\_eq\_projection\_test
  - rel\_eq\_projection.c, 166
  - rel\_eq\_projection.h, 170
- AK\_rel\_eq\_remove\_duplicates
  - rel\_eq\_projection.c, 166
  - rel\_eq\_projection.h, 170
- AK\_rel\_eq\_selection
  - rel\_eq\_selection.c, 173
  - rel\_eq\_selection.h, 177
- AK\_rel\_eq\_selection\_test
  - rel\_eq\_selection.c, 173
  - rel\_eq\_selection.h, 177
- AK\_rel\_eq\_share\_attributes
  - rel\_eq\_selection.c, 173
  - rel\_eq\_selection.h, 177
- AK\_rel\_eq\_split\_condition
  - rel\_eq\_selection.c, 174
  - rel\_eq\_selection.h, 178
- AK\_release\_locks
  - transaction.c, 293
  - transaction.h, 305
- AK\_remove\_all\_users\_from\_group
  - privileges.c, 271
- AK\_remove\_transaction\_thread
  - transaction.c, 293
  - transaction.h, 305
- AK\_remove\_user\_from\_all\_groups
  - privileges.c, 271
- AK\_rename
  - table.c, 123
  - table.h, 132
- AK\_replace\_wild\_card
  - expression\_check.c, 194
- AK\_results, 18
- AK\_revoke\_all\_privileges\_group
  - privileges.c, 271
- AK\_revoke\_all\_privileges\_user
  - privileges.c, 272
- AK\_revoke\_privilege\_group
  - privileges.c, 272
- AK\_revoke\_privilege\_user
  - privileges.c, 272
- AK\_search\_empty\_link\_for\_hook
  - transaction.c, 293
  - transaction.h, 305
- AK\_search\_existing\_link\_for\_hook
  - transaction.c, 293
  - transaction.h, 306
- AK\_search\_lock\_entry\_list\_by\_key
  - transaction.c, 294
  - transaction.h, 306
- AK\_search\_unsorted
  - aggregation.c, 186
  - filesearch.c, 77
  - filesearch.h, 79
- AK\_select
  - select.c, 274
- AK\_select\_test
  - select.c, 275
- AK\_selection
  - reference.h, 250
  - selection.c, 214
  - selection.h, 216
- AK\_sequence\_add
  - sequence.c, 217
  - sequence.h, 220
- AK\_sequence\_current\_value
  - sequence.c, 217
  - sequence.h, 220
- AK\_sequence\_get\_id
  - sequence.c, 217
  - sequence.h, 220
- AK\_sequence\_modify
  - sequence.c, 217
  - sequence.h, 220
- AK\_sequence\_next\_value
  - sequence.c, 218
  - sequence.h, 221
- AK\_sequence\_remove
  - sequence.c, 218
  - sequence.h, 221
- AK\_sequence\_rename
  - sequence.c, 218
  - sequence.h, 221
- AK\_sequence\_test
  - sequence.c, 219
  - sequence.h, 222
- AK\_set\_check\_constraint
  - check\_constraint.c, 233
  - check\_constraint.h, 235
- AK\_set\_constraint\_between

- between.c, [230](#)
  - between.h, [231](#)
- AK\_set\_constraint\_not\_null
  - nnull.c, [239](#)
  - nnull.h, [240](#)
- AK\_split\_path\_file
  - blobs.c, [62](#)
  - blobs.h, [65](#)
- AK\_table\_empty
  - table.c, [124](#)
  - table.h, [133](#)
- AK\_table\_exist
  - table.c, [124](#)
- AK\_table\_test
  - table.c, [124](#)
  - table.h, [133](#)
- AK\_temp\_create\_table
  - projection.c, [209](#)
  - projection.h, [213](#)
- AK\_theta\_join
  - theta\_join.c, [223](#)
  - theta\_join.h, [225](#)
- AK\_thread\_safe\_block\_access\_test
  - dbman.c, [43](#)
  - dbman.h, [58](#)
- AK\_transaction\_finished
  - transaction.c, [294](#)
  - transaction.h, [306](#)
- AK\_transaction\_manager
  - transaction.c, [294](#)
  - transaction.h, [306](#)
- AK\_transaction\_register\_observer
  - transaction.c, [295](#)
  - transaction.h, [307](#)
- AK\_transaction\_unregister\_observer
  - transaction.c, [295](#)
  - transaction.h, [307](#)
- AK\_trigger\_add
  - trigger.c, [276](#)
  - trigger.h, [279](#)
- AK\_trigger\_edit
  - trigger.c, [276](#)
  - trigger.h, [280](#)
- AK\_trigger\_get\_conditions
  - trigger.c, [276](#)
  - trigger.h, [280](#)
- AK\_trigger\_get\_id
  - trigger.c, [277](#)
  - trigger.h, [280](#)
- AK\_trigger\_remove\_by\_name
  - trigger.c, [277](#)
  - trigger.h, [281](#)
- AK\_trigger\_remove\_by\_obj\_id
  - trigger.c, [277](#)
  - trigger.h, [281](#)
- AK\_trigger\_rename
  - trigger.c, [278](#)
  - trigger.h, [281](#)
- AK\_trigger\_save\_conditions
  - trigger.c, [278](#)
  - trigger.h, [281](#)
- AK\_trigger\_test
  - trigger.c, [278](#)
  - trigger.h, [282](#)
- AK\_tuple\_dict, [18](#)
- AK\_tuple\_to\_string
  - table.c, [124](#)
  - table.h, [133](#)
- AK\_union
  - union.c, [226](#)
  - union.h, [227](#)
- AK\_unique\_test
  - unique.c, [252](#)
  - unique.h, [254](#)
- AK\_update
  - bitmap.c, [88](#)
  - bitmap.h, [93](#)
- AK\_user\_add
  - privileges.c, [273](#)
- AK\_user\_get\_id
  - privileges.c, [273](#)
- AK\_user\_remove\_by\_name
  - privileges.c, [273](#)
- AK\_user\_rename
  - privileges.c, [274](#)
- AK\_view\_add
  - view.c, [283](#)
- AK\_view\_change\_query
  - view.c, [284](#)
- AK\_view\_remove\_by\_name
  - view.c, [284](#)
- AK\_view\_remove\_by\_obj\_id
  - view.c, [284](#)
- AK\_view\_rename
  - view.c, [285](#)
- AK\_view\_test
  - view.c, [285](#)
- AK\_write\_block
  - dbman.c, [43](#)
  - dbman.h, [58](#)
- AK\_write\_block\_for\_testing
  - dbman.c, [43](#)
  - dbman.h, [58](#)
- aggregation.c
  - AK\_agg\_input\_add, [184](#)
  - AK\_agg\_input\_add\_to\_beginning, [184](#)
  - AK\_agg\_input\_fix, [184](#)
  - AK\_agg\_input\_init, [185](#)
  - AK\_aggregation, [185](#)
  - AK\_header\_size, [186](#)
  - AK\_search\_unsorted, [186](#)
  - Ak\_aggregation\_test, [186](#)
- aggregation.h
  - AK\_agg\_input\_add, [188](#)
  - AK\_agg\_input\_add\_to\_beginning, [188](#)
  - AK\_agg\_input\_fix, [189](#)

- AK\_agg\_input\_init, 189
- AK\_aggregation, 189
- AK\_header\_size, 190
- Ak\_aggregation\_test, 190
- Ak\_Delete\_All\_elementsAd
  - index.c, 106
  - index.h, 112
- Ak\_Delete\_elementAd
  - index.c, 106
  - index.h, 112
- Ak\_Get\_First\_elementAd
  - index.c, 106
  - index.h, 112
- Ak\_Get\_Last\_elementAd
  - index.c, 108
  - index.h, 113
- Ak\_Get\_Next\_elementAd
  - index.c, 108
  - index.h, 114
- Ak\_Get\_Position\_Of\_elementAd
  - index.c, 108
  - index.h, 114
- Ak\_Get\_Previous\_elementAd
  - index.c, 109
  - index.h, 114
- Ak\_If\_ExistOp
  - bitmap.c, 87
  - bitmap.h, 92
- Ak\_InitializelistAd
  - index.c, 109
  - index.h, 115
- Ak\_Insert\_New\_Element
  - fileio.c, 67
  - fileio.h, 72
  - reference.h, 247
- Ak\_Insert\_New\_Element\_For\_Update
  - fileio.c, 68
  - fileio.h, 72
  - reference.h, 248
- Ak\_Insert\_NewelementAd
  - index.c, 110
  - index.h, 115
- Ak\_aggregation\_test
  - aggregation.c, 186
  - aggregation.h, 190
- Ak\_bitmap\_test
  - bitmap.c, 85
  - bitmap.h, 90
- Ak\_check\_constraint\_name
  - constraint\_names.c, 236
  - constraint\_names.h, 237
- Ak\_check\_regex\_expression
  - expression\_check.c, 194
  - expression\_check.h, 196
- Ak\_check\_regex\_operator\_expression
  - expression\_check.c, 194
  - expression\_check.h, 196
- Ak\_constraint\_between\_test
  - between.c, 228
  - between.h, 231
- Ak\_create\_Index
  - bitmap.c, 85
  - bitmap.h, 90
- Ak\_delete\_row
  - fileio.c, 66
  - fileio.h, 71
  - reference.h, 246
- Ak\_delete\_row\_by\_id
  - fileio.c, 66
  - fileio.h, 71
- Ak\_delete\_row\_from\_block
  - fileio.c, 67
  - fileio.h, 71
- Ak\_delete\_update\_segment
  - fileio.c, 67
  - fileio.h, 71
- Ak\_files\_test
  - files.c, 74
  - files.h, 76
- Ak\_filesearch\_test
  - filesearch.c, 77
  - filesearch.h, 79
- Ak\_get\_Attribute
  - bitmap.c, 86
  - bitmap.h, 91
- Ak\_get\_header\_number
  - filesort.h, 81
- Ak\_get\_nth\_main\_bucket\_add
  - hash.c, 99
  - hash.h, 103
- Ak\_get\_num\_of\_tuples
  - filesort.h, 81
- Ak\_get\_total\_headers
  - filesort.h, 81
- Ak\_hash\_test
  - hash.c, 99
  - hash.h, 104
- Ak\_id\_test
  - id.c, 82
  - id.h, 83
- Ak\_insert\_bucket\_to\_block
  - hash.c, 99
  - hash.h, 104
- Ak\_insert\_row
  - fileio.c, 68
  - fileio.h, 72
  - reference.h, 248
- Ak\_insert\_row\_to\_block
  - fileio.c, 68
  - fileio.h, 73
- Ak\_op\_difference\_test
  - difference.c, 191
  - difference.h, 192
- Ak\_op\_intersect\_test
  - intersect.c, 197
  - intersect.h, 198

- Ak\_print\_Att\_Test
  - bitmap.c, [87](#)
  - bitmap.h, [92](#)
- Ak\_print\_Header\_Test
  - bitmap.c, [87](#)
  - bitmap.h, [92](#)
- Ak\_set\_constraint\_unique
  - unique.c, [252](#)
  - unique.h, [254](#)
- Ak\_update\_bucket\_in\_block
  - hash.c, [100](#)
  - hash.h, [104](#)
- Ak\_update\_row
  - fileio.c, [69](#)
  - fileio.h, [73](#)
  - reference.h, [250](#)
- Ak\_update\_row\_from\_block
  - fileio.c, [69](#)
  - fileio.h, [73](#)
- Ak\_write\_block
  - bitmap.c, [88](#)
  - bitmap.h, [93](#)
- archive\_log.h
  - AK\_archive\_log, [179](#)
  - AK\_empty\_archive\_log, [179](#)
  - AK\_get\_timestamp, [179](#)
- between.c
  - AK\_delete\_constraint\_between, [228](#)
  - AK\_find\_table\_address, [229](#)
  - AK\_print\_constraints, [229](#)
  - AK\_read\_constraint\_between, [229](#)
  - AK\_set\_constraint\_between, [230](#)
  - Ak\_constraint\_between\_test, [228](#)
- between.h
  - AK\_find\_table\_address, [231](#)
  - AK\_read\_constraint\_between, [231](#)
  - AK\_set\_constraint\_between, [231](#)
  - Ak\_constraint\_between\_test, [231](#)
- bitmap.c
  - AK\_add\_to\_bitmap\_index, [85](#)
  - AK\_create\_Index\_Table, [86](#)
  - AK\_delete\_bitmap\_index, [86](#)
  - AK\_get\_Attribute, [86](#)
  - AK\_update, [88](#)
  - Ak\_If\_ExistOp, [87](#)
  - Ak\_bitmap\_test, [85](#)
  - Ak\_create\_Index, [85](#)
  - Ak\_get\_Attribute, [86](#)
  - Ak\_print\_Att\_Test, [87](#)
  - Ak\_print\_Header\_Test, [87](#)
  - Ak\_write\_block, [88](#)
- bitmap.h
  - AK\_add\_to\_bitmap\_index, [89](#)
  - AK\_create\_Index\_Table, [90](#)
  - AK\_delete\_bitmap\_index, [91](#)
  - AK\_get\_Attribute, [91](#)
  - AK\_update, [93](#)
  - Ak\_If\_ExistOp, [92](#)
  - Ak\_bitmap\_test, [90](#)
  - Ak\_create\_Index, [90](#)
  - Ak\_get\_Attribute, [91](#)
  - Ak\_print\_Att\_Test, [92](#)
  - Ak\_print\_Header\_Test, [92](#)
  - Ak\_write\_block, [93](#)
- blobs.c
  - AK\_GUID, [61](#)
  - AK\_check\_folder\_blobs, [60](#)
  - AK\_concat, [60](#)
  - AK\_folder\_exists, [60](#)
  - AK\_lo\_export, [61](#)
  - AK\_lo\_import, [61](#)
  - AK\_lo\_test, [61](#)
  - AK\_lo\_unlink, [61](#)
  - AK\_mkdir, [62](#)
  - AK\_split\_path\_file, [62](#)
- blobs.h
  - AK\_GUID, [64](#)
  - AK\_check\_folder\_blobs, [63](#)
  - AK\_concat, [63](#)
  - AK\_folder\_exists, [64](#)
  - AK\_lo\_export, [64](#)
  - AK\_lo\_import, [64](#)
  - AK\_lo\_test, [64](#)
  - AK\_lo\_unlink, [65](#)
  - AK\_mkdir, [65](#)
  - AK\_split\_path\_file, [65](#)
- blocktable, [19](#)
- btree.c
  - AK\_btree\_create, [94](#)
  - AK\_btree\_search\_delete, [94](#)
- btree.h
  - AK\_btree\_create, [95](#)
  - AK\_btree\_search\_delete, [95](#)
- btree\_node, [19](#)
- bucket\_elem, [20](#)
- CHAR\_IN\_LINE
  - dbman.h, [47](#)
- check\_constraint.c
  - AK\_check\_constraint, [232](#)
  - AK\_check\_constraint\_test, [233](#)
  - AK\_set\_check\_constraint, [233](#)
  - condition\_passed, [233](#)
- check\_constraint.h
  - AK\_check\_constraint, [234](#)
  - AK\_check\_constraint\_test, [235](#)
  - AK\_set\_check\_constraint, [235](#)
  - condition\_passed, [235](#)
- condition\_passed
  - check\_constraint.c, [233](#)
  - check\_constraint.h, [235](#)
- constraint\_names.c
  - AK\_constraint\_names\_test, [236](#)
  - Ak\_check\_constraint\_name, [236](#)
- constraint\_names.h
  - AK\_constraint\_names\_test, [237](#)
  - Ak\_check\_constraint\_name, [237](#)

- cost\_eval\_t, 20
- create\_header\_test
  - test.c, 135
  - test.h, 138
- db
  - dbman.h, 59
- db\_file\_size
  - dbman.h, 59
- dbman.c
  - AK\_allocate\_block\_activity\_modes, 33
  - AK\_allocate\_blocks, 33
  - AK\_allocationtable\_dump, 33
  - AK\_blocktable\_dump, 34
  - AK\_blocktable\_flush, 34
  - AK\_blocktable\_get, 34
  - AK\_copy\_header, 34
  - AK\_create\_header, 35
  - AK\_delete\_block, 35
  - AK\_delete\_extent, 35
  - AK\_delete\_segment, 36
  - AK\_get\_allocation\_set, 36
  - AK\_get\_extent, 36
  - AK\_increase\_extent, 37
  - AK\_init\_allocation\_table, 37
  - AK\_init\_block, 38
  - AK\_init\_db\_file, 38
  - AK\_init\_disk\_manager, 38
  - AK\_init\_system\_catalog, 38
  - AK\_init\_system\_tables\_catalog, 39
  - AK\_insert\_entry, 40
  - AK\_memset\_int, 40
  - AK\_new\_extent, 41
  - AK\_new\_segment, 41
  - AK\_print\_block, 42
  - AK\_read\_block, 42
  - AK\_read\_block\_for\_testing, 42
  - AK\_register\_system\_tables, 42
  - AK\_thread\_safe\_block\_access\_test, 43
  - AK\_write\_block, 43
  - AK\_write\_block\_for\_testing, 43
  - fsize, 44
- dbman.h
  - AK\_allocate\_blocks, 48
  - AK\_allocation\_set\_mode, 48
  - AK\_allocationbit, 59
  - AK\_allocationtable\_dump, 48
  - AK\_blocktable\_dump, 48
  - AK\_blocktable\_flush, 48
  - AK\_blocktable\_get, 49
  - AK\_copy\_header, 49
  - AK\_create\_header, 49
  - AK\_delete\_block, 50
  - AK\_delete\_extent, 50
  - AK\_delete\_segment, 50
  - AK\_get\_allocation\_set, 51
  - AK\_get\_extent, 51
  - AK\_increase\_extent, 51
  - AK\_init\_allocation\_table, 52
  - AK\_init\_block, 52
  - AK\_init\_db\_file, 52
  - AK\_init\_disk\_manager, 53
  - AK\_init\_system\_catalog, 53
  - AK\_init\_system\_tables\_catalog, 53
  - AK\_insert\_entry, 54
  - AK\_memset\_int, 55
  - AK\_new\_extent, 55
  - AK\_new\_segment, 56
  - AK\_print\_block, 56
  - AK\_read\_block, 56
  - AK\_read\_block\_for\_testing, 57
  - AK\_register\_system\_tables, 57
  - AK\_thread\_safe\_block\_access\_test, 58
  - AK\_write\_block, 58
  - AK\_write\_block\_for\_testing, 58
  - CHAR\_IN\_LINE, 47
  - db, 59
  - db\_file\_size, 59
  - fsize, 58
  - MAX\_BLOCK\_INIT\_NUM, 47
- difference.c
  - AK\_difference, 191
  - Ak\_op\_difference\_test, 191
- difference.h
  - AK\_difference, 192
  - Ak\_op\_difference\_test, 192
- dm/dbman.c, 31
- dm/dbman.h, 44
- drop.c
  - AK\_drop, 255
  - AK\_drop\_help\_function, 255
  - AK\_drop\_test, 256
  - AK\_if\_exist, 256
  - system\_catalog, 256
- drop.h
  - AK\_drop, 257
  - AK\_drop\_test, 257
  - AK\_if\_exist, 257
- drop\_arguments, 21
- expression\_check.c
  - AK\_check\_arithmetic\_statement, 193
  - AK\_check\_if\_row\_satisfies\_expression, 193
  - AK\_replace\_wild\_card, 194
  - Ak\_check\_regex\_expression, 194
  - Ak\_check\_regex\_operator\_expression, 194
- expression\_check.h
  - AK\_check\_arithmetic\_statement, 195
  - AK\_check\_if\_row\_satisfies\_expression, 195
  - Ak\_check\_regex\_expression, 196
  - Ak\_check\_regex\_operator\_expression, 196
- file/blobs.c, 59
- file/blobs.h, 62
- file/fileio.c, 65
- file/fileio.h, 70
- file/files.c, 74
- file/files.h, 75

- file/filesearch.c, [77](#)
- file/filesearch.h, [78](#)
- file/filesort.h, [80](#)
- file/id.c, [82](#)
- file/id.h, [83](#)
- file/idx/bitmap.c, [84](#)
- file/idx/bitmap.h, [88](#)
- file/idx/btree.c, [93](#)
- file/idx/btree.h, [94](#)
- file/idx/hash.c, [96](#)
- file/idx/hash.h, [100](#)
- file/idx/index.c, [105](#)
- file/idx/index.h, [111](#)
- file/table.c, [116](#)
- file/table.h, [125](#)
- file/test.c, [134](#)
- file/test.h, [137](#)
- fileio.c
  - Ak\_Insert\_New\_Element, [67](#)
  - Ak\_Insert\_New\_Element\_For\_Update, [68](#)
  - Ak\_delete\_row, [66](#)
  - Ak\_delete\_row\_by\_id, [66](#)
  - Ak\_delete\_row\_from\_block, [67](#)
  - Ak\_delete\_update\_segment, [67](#)
  - Ak\_insert\_row, [68](#)
  - Ak\_insert\_row\_to\_block, [68](#)
  - Ak\_update\_row, [69](#)
  - Ak\_update\_row\_from\_block, [69](#)
- fileio.h
  - Ak\_Insert\_New\_Element, [72](#)
  - Ak\_Insert\_New\_Element\_For\_Update, [72](#)
  - Ak\_delete\_row, [71](#)
  - Ak\_delete\_row\_by\_id, [71](#)
  - Ak\_delete\_row\_from\_block, [71](#)
  - Ak\_delete\_update\_segment, [71](#)
  - Ak\_insert\_row, [72](#)
  - Ak\_insert\_row\_to\_block, [73](#)
  - Ak\_update\_row, [73](#)
  - Ak\_update\_row\_from\_block, [73](#)
- files.c
  - AK\_initialize\_new\_index\_segment, [74](#)
  - AK\_initialize\_new\_segment, [75](#)
  - Ak\_files\_test, [74](#)
- files.h
  - AK\_initialize\_new\_index\_segment, [76](#)
  - AK\_initialize\_new\_segment, [76](#)
  - Ak\_files\_test, [76](#)
- filesearch.c
  - AK\_deallocate\_search\_result, [77](#)
  - AK\_search\_unsorted, [77](#)
  - Ak\_filesearch\_test, [77](#)
- filesearch.h
  - AK\_deallocate\_search\_result, [79](#)
  - AK\_search\_unsorted, [79](#)
  - Ak\_filesearch\_test, [79](#)
- filesort.h
  - Ak\_block\_sort, [81](#)
  - Ak\_get\_header\_number, [81](#)
  - Ak\_get\_num\_of\_tuples, [81](#)
  - Ak\_get\_total\_headers, [81](#)
- fsz
  - dbman.c, [44](#)
  - dbman.h, [58](#)
- function.c
  - AK\_check\_function\_arguments, [259](#)
  - AK\_check\_function\_arguments\_type, [259](#)
  - AK\_function\_add, [259](#)
  - AK\_function\_arguments\_add, [259](#)
  - AK\_function\_arguments\_remove\_by\_obj\_id, [260](#)
  - AK\_function\_change\_return\_type, [260](#)
  - AK\_function\_remove\_by\_name, [260](#)
  - AK\_function\_remove\_by\_obj\_id, [261](#)
  - AK\_function\_rename, [261](#)
  - AK\_function\_test, [261](#)
  - AK\_get\_function\_obj\_id, [262](#)
- function.h
  - AK\_check\_function\_arguments, [263](#)
  - AK\_check\_function\_arguments\_type, [263](#)
  - AK\_function\_add, [263](#)
  - AK\_function\_arguments\_add, [264](#)
  - AK\_function\_arguments\_remove\_by\_obj\_id, [264](#)
  - AK\_function\_change\_return\_type, [264](#)
  - AK\_function\_remove\_by\_name, [265](#)
  - AK\_function\_remove\_by\_obj\_id, [265](#)
  - AK\_function\_rename, [265](#)
  - AK\_function\_test, [266](#)
  - AK\_get\_function\_obj\_id, [266](#)
- get\_column\_test
  - test.c, [135](#)
  - test.h, [138](#)
- get\_row\_attr\_data
  - table.c, [125](#)
  - table.h, [133](#)
- get\_row\_test
  - test.c, [136](#)
  - test.h, [138](#)
- grandfailure
  - recovery.c, [182](#)
- handle\_transaction\_notify
  - transaction.c, [295](#)
  - transaction.h, [307](#)
- hash.c
  - AK\_change\_hash\_info, [96](#)
  - AK\_create\_hash\_index, [97](#)
  - AK\_delete\_in\_hash\_index, [97](#)
  - AK\_elem\_hash\_value, [97](#)
  - AK\_find\_delete\_in\_hash\_index, [98](#)
  - AK\_find\_in\_hash\_index, [98](#)
  - AK\_get\_hash\_info, [98](#)
  - AK\_insert\_in\_hash\_index, [99](#)
  - Ak\_get\_nth\_main\_bucket\_add, [99](#)
  - Ak\_hash\_test, [99](#)
  - Ak\_insert\_bucket\_to\_block, [99](#)
  - Ak\_update\_bucket\_in\_block, [100](#)
- hash.h

- AK\_change\_hash\_info, 101
- AK\_create\_hash\_index, 101
- AK\_delete\_in\_hash\_index, 102
- AK\_elem\_hash\_value, 102
- AK\_find\_delete\_in\_hash\_index, 102
- AK\_find\_in\_hash\_index, 103
- AK\_get\_hash\_info, 103
- AK\_insert\_in\_hash\_index, 104
- Ak\_get\_nth\_main\_bucket\_add, 103
- Ak\_hash\_test, 104
- Ak\_insert\_bucket\_to\_block, 104
- Ak\_update\_bucket\_in\_block, 104
- hash\_bucket, 21
- hash\_info, 21
- id.c
  - AK\_get\_id, 82
  - AK\_get\_table\_id, 82
  - Ak\_id\_test, 82
- id.h
  - AK\_get\_id, 83
  - Ak\_id\_test, 83
- index.c
  - AK\_get\_index\_header, 107
  - AK\_get\_index\_num\_records, 107
  - AK\_get\_index\_tuple, 107
  - AK\_index\_table\_exist, 109
  - AK\_index\_test, 109
  - AK\_num\_index\_attr, 110
  - AK\_print\_index\_table, 110
  - Ak\_Delete\_All\_elementsAd, 106
  - Ak\_Delete\_elementAd, 106
  - Ak\_Get\_First\_elementAd, 106
  - Ak\_Get\_Last\_elementAd, 108
  - Ak\_Get\_Next\_elementAd, 108
  - Ak\_Get\_Position\_Of\_elementAd, 108
  - Ak\_Get\_Previous\_elementAd, 109
  - Ak\_InitializelistAd, 109
  - Ak\_Insert\_NewelementAd, 110
- index.h
  - AK\_get\_index\_num\_records, 113
  - AK\_get\_index\_tuple, 113
  - AK\_index\_table\_exist, 115
  - AK\_index\_test, 115
  - AK\_num\_index\_attr, 116
  - AK\_print\_index\_table, 116
  - Ak\_Delete\_All\_elementsAd, 112
  - Ak\_Delete\_elementAd, 112
  - Ak\_Get\_First\_elementAd, 112
  - Ak\_Get\_Last\_elementAd, 113
  - Ak\_Get\_Next\_elementAd, 114
  - Ak\_Get\_Position\_Of\_elementAd, 114
  - Ak\_Get\_Previous\_elementAd, 114
  - Ak\_InitializelistAd, 115
  - Ak\_Insert\_NewelementAd, 115
- insert\_data\_test
  - test.c, 136
  - test.h, 139
- intersect.c
  - AK\_intersect, 197
  - Ak\_op\_intersect\_test, 197
- intersect.h
  - AK\_intersect, 198
  - Ak\_op\_intersect\_test, 198
- intersect\_attr, 22
- list\_structure\_ad, 22
- list\_structure\_add, 23
- MAX\_BLOCK\_INIT\_NUM
  - dbman.h, 47
- main\_bucket, 23
- memoman.c
  - AK\_cache\_AK\_malloc, 141
  - AK\_cache\_block, 141
  - AK\_cache\_result, 141
  - AK\_find\_AK\_free\_space, 141
  - AK\_find\_available\_result\_block, 142
  - AK\_flush\_cache, 142
  - AK\_generate\_result\_id, 142
  - AK\_get\_block, 142
  - AK\_get\_index\_addresses, 143
  - AK\_get\_index\_segment\_addresses, 143
  - AK\_get\_segment\_addresses, 143
  - AK\_get\_table\_addresses, 144
  - AK\_init\_new\_extent, 144
  - AK\_mem\_block\_modify, 144
  - AK\_memoman\_init, 145
  - AK\_query\_mem\_AK\_malloc, 145
  - AK\_redo\_log\_AK\_malloc, 145
  - AK\_refresh\_cache, 145
- memoman.h
  - AK\_cache\_AK\_malloc, 147
  - AK\_cache\_block, 148
  - AK\_cache\_result, 148
  - AK\_find\_AK\_free\_space, 148
  - AK\_find\_available\_result\_block, 148
  - AK\_flush\_cache, 149
  - AK\_generate\_result\_id, 149
  - AK\_get\_block, 149
  - AK\_get\_index\_addresses, 149
  - AK\_get\_index\_segment\_addresses, 150
  - AK\_get\_segment\_addresses, 150
  - AK\_get\_table\_addresses, 150
  - AK\_init\_new\_extent, 151
  - AK\_mem\_block\_modify, 151
  - AK\_memoman\_init, 151
  - AK\_query\_mem\_AK\_malloc, 151
  - AK\_redo\_log\_AK\_malloc, 152
  - AK\_refresh\_cache, 152
- memoryAddresses, 23
- mm/memoman.c, 140
- mm/memoman.h, 146
- nat\_join.c
  - AK\_copy\_blocks\_join, 199
  - AK\_create\_join\_block\_header, 199
  - AK\_join, 200



- AK\_merge\_block\_join, 200
- AK\_op\_join\_test, 200
- nat\_join.h
  - AK\_copy\_blocks\_join, 201
  - AK\_create\_join\_block\_header, 202
  - AK\_join, 202
  - AK\_merge\_block\_join, 202
  - AK\_op\_join\_test, 203
- nnull.c
  - AK\_delete\_constraint\_not\_null, 238
  - AK\_null\_test, 238
  - AK\_read\_constraint\_not\_null, 238
  - AK\_set\_constraint\_not\_null, 239
- nnull.h
  - AK\_delete\_constraint\_not\_null, 240
  - AK\_null\_test, 240
  - AK\_read\_constraint\_not\_null, 240
  - AK\_set\_constraint\_not\_null, 240
- NoticeType
  - transaction.h, 298
- observable\_transaction, 24
- observable\_transaction\_struct, 24
- observer\_lock, 24
- opti/query\_optimization.c, 152
- opti/query\_optimization.h, 154
- opti/rel\_eq\_assoc.c, 156
- opti/rel\_eq\_assoc.h, 158
- opti/rel\_eq\_comut.c, 159
- opti/rel\_eq\_comut.h, 161
- opti/rel\_eq\_projection.c, 163
- opti/rel\_eq\_projection.h, 167
- opti/rel\_eq\_selection.c, 171
- opti/rel\_eq\_selection.h, 175
- privileges.c
  - AK\_add\_user\_to\_group, 267
  - AK\_check\_group\_privilege, 268
  - AK\_check\_privilege, 268
  - AK\_check\_user\_privilege, 268
  - AK\_grant\_privilege\_group, 269
  - AK\_grant\_privilege\_user, 269
  - AK\_group\_add, 269
  - AK\_group\_get\_id, 270
  - AK\_group\_remove\_by\_name, 270
  - AK\_group\_rename, 270
  - AK\_privileges\_test, 270
  - AK\_remove\_all\_users\_from\_group, 271
  - AK\_remove\_user\_from\_all\_groups, 271
  - AK\_revoke\_all\_privileges\_group, 271
  - AK\_revoke\_all\_privileges\_user, 272
  - AK\_revoke\_privilege\_group, 272
  - AK\_revoke\_privilege\_user, 272
  - AK\_user\_add, 273
  - AK\_user\_get\_id, 273
  - AK\_user\_remove\_by\_name, 273
  - AK\_user\_rename, 274
- product.c
  - AK\_op\_product\_test, 203
- AK\_product, 203
- product.h
  - AK\_op\_product\_test, 204
  - AK\_product, 205
- projection.c
  - AK\_copy\_block\_projection, 206
  - AK\_create\_block\_header, 206
  - AK\_create\_header\_name, 206
  - AK\_determine\_header\_type, 208
  - AK\_get\_operator, 208
  - AK\_op\_projection\_test, 208
  - AK\_perform\_operatrion, 208
  - AK\_projection, 209
  - AK\_temp\_create\_table, 209
  - removeSubstring, 209
- projection.h
  - AK\_copy\_block\_projection, 211
  - AK\_create\_block\_header, 211
  - AK\_create\_header\_name, 211
  - AK\_determine\_header\_type, 211
  - AK\_get\_operator, 212
  - AK\_op\_projection\_test, 212
  - AK\_perform\_operatrion, 212
  - AK\_projection, 212
  - AK\_temp\_create\_table, 213
  - removeSubstring, 213
- query\_optimization.c
  - AK\_execute\_rel\_eq, 153
  - AK\_print\_optimized\_query, 153
  - AK\_query\_optimization, 153
  - AK\_query\_optimization\_test, 154
- query\_optimization.h
  - AK\_execute\_rel\_eq, 155
  - AK\_print\_optimized\_query, 155
  - AK\_query\_optimization, 155
  - AK\_query\_optimization\_test, 156
- REF\_TYPE\_NO\_ACTION
  - reference.h, 246
- rec/archive\_log.h, 179
- rec/recovery.c, 180
- rec/redo\_log.c, 182
- recovery.c
  - AK\_recover\_archive\_log, 180
  - AK\_recover\_operation, 181
  - AK\_recovery\_insert\_row, 181
  - AK\_recovery\_test, 181
  - AK\_recovery\_tokenize, 181
  - grandfailure, 182
- redo\_log.c
  - AK\_add\_to\_redolog, 182
  - AK\_check\_attributes, 183
  - AK\_printout\_redolog, 183
- reference.c
  - AK\_add\_reference, 242
  - AK\_get\_reference, 242
  - AK\_reference\_check\_attribute, 242
  - AK\_reference\_check\_entry, 243



- AK\_reference\_check\_if\_update\_needed, 243
- AK\_reference\_check\_restricion, 243
- AK\_reference\_test, 244
- AK\_reference\_update, 244
- reference.h
  - AK\_add\_reference, 246
  - AK\_get\_reference, 247
  - AK\_initialize\_new\_segment, 247
  - AK\_reference\_check\_attribute, 248
  - AK\_reference\_check\_entry, 249
  - AK\_reference\_check\_if\_update\_needed, 249
  - AK\_reference\_check\_restricion, 249
  - AK\_reference\_test, 250
  - AK\_reference\_update, 250
  - AK\_selection, 250
  - Ak\_Insert\_New\_Element, 247
  - Ak\_Insert\_New\_Element\_For\_Update, 248
  - Ak\_delete\_row, 246
  - Ak\_insert\_row, 248
  - Ak\_update\_row, 250
  - REF\_TYPE\_NO\_ACTION, 246
- rel/aggregation.c, 183
- rel/aggregation.h, 187
- rel/difference.c, 190
- rel/difference.h, 191
- rel/expression\_check.c, 192
- rel/expression\_check.h, 195
- rel/intersect.c, 196
- rel/intersect.h, 197
- rel/nat\_join.c, 198
- rel/nat\_join.h, 201
- rel/product.c, 203
- rel/product.h, 204
- rel/projection.c, 205
- rel/projection.h, 210
- rel/selection.c, 213
- rel/selection.h, 215
- rel/sequence.c, 216
- rel/sequence.h, 219
- rel/theta\_join.c, 222
- rel/theta\_join.h, 224
- rel/union.c, 226
- rel/union.h, 227
- rel\_eq\_assoc.c
  - AK\_compare, 157
  - AK\_print\_rel\_eq\_assoc, 157
  - AK\_rel\_eq\_assoc, 157
  - AK\_rel\_eq\_assoc\_test, 157
- rel\_eq\_assoc.h
  - AK\_compare, 158
  - AK\_print\_rel\_eq\_assoc, 159
  - AK\_rel\_eq\_assoc, 159
  - AK\_rel\_eq\_assoc\_test, 159
- rel\_eq\_comut.c
  - AK\_print\_rel\_eq\_comut, 160
  - AK\_rel\_eq\_commute\_with\_theta\_join, 160
  - AK\_rel\_eq\_comut, 160
  - AK\_rel\_eq\_comut\_test, 161
- rel\_eq\_comut.h
  - AK\_print\_rel\_eq\_comut, 162
  - AK\_rel\_eq\_commute\_with\_theta\_join, 162
  - AK\_rel\_eq\_comut, 162
  - AK\_rel\_eq\_comut\_test, 162
- rel\_eq\_projection.c
  - AK\_print\_rel\_eq\_projection, 163
  - AK\_rel\_eq\_can\_commute, 164
  - AK\_rel\_eq\_collect\_cond\_attributes, 164
  - AK\_rel\_eq\_get\_attributes, 164
  - AK\_rel\_eq\_is\_subset, 165
  - AK\_rel\_eq\_projection, 165
  - AK\_rel\_eq\_projection\_attributes, 166
  - AK\_rel\_eq\_projection\_test, 166
  - AK\_rel\_eq\_remove\_duplicates, 166
- rel\_eq\_projection.h
  - AK\_print\_rel\_eq\_projection, 167
  - AK\_rel\_eq\_can\_commute, 168
  - AK\_rel\_eq\_collect\_cond\_attributes, 168
  - AK\_rel\_eq\_get\_attributes, 168
  - AK\_rel\_eq\_is\_subset, 169
  - AK\_rel\_eq\_projection, 169
  - AK\_rel\_eq\_projection\_attributes, 170
  - AK\_rel\_eq\_projection\_test, 170
  - AK\_rel\_eq\_remove\_duplicates, 170
- rel\_eq\_selection.c
  - AK\_print\_rel\_eq\_selection, 171
  - AK\_rel\_eq\_cond\_attributes, 172
  - AK\_rel\_eq\_get\_attributes\_char, 172
  - AK\_rel\_eq\_is\_attr\_subset, 172
  - AK\_rel\_eq\_selection, 173
  - AK\_rel\_eq\_selection\_test, 173
  - AK\_rel\_eq\_share\_attributes, 173
  - AK\_rel\_eq\_split\_condition, 174
- rel\_eq\_selection.h
  - AK\_print\_rel\_eq\_selection, 175
  - AK\_rel\_eq\_cond\_attributes, 176
  - AK\_rel\_eq\_get\_attributes\_char, 176
  - AK\_rel\_eq\_is\_attr\_subset, 176
  - AK\_rel\_eq\_selection, 177
  - AK\_rel\_eq\_selection\_test, 177
  - AK\_rel\_eq\_share\_attributes, 177
  - AK\_rel\_eq\_split\_condition, 178
- removeSubstring
  - projection.c, 209
  - projection.h, 213
- root\_info, 25
- search\_params, 25
- search\_result, 26
- select.c
  - AK\_select, 274
  - AK\_select\_test, 275
- selection.c
  - AK\_op\_selection\_test, 214
  - AK\_op\_selection\_test2, 214
  - AK\_op\_selection\_test\_redolog, 214
  - AK\_selection, 214
- selection.h

- AK\_op\_selection\_test, 215
- AK\_op\_selection\_test2, 215
- AK\_op\_selection\_test\_redolog, 215
- AK\_selection, 216
- selection\_test
  - test.c, 136
  - test.h, 139
- sequence.c
  - AK\_sequence\_add, 217
  - AK\_sequence\_current\_value, 217
  - AK\_sequence\_get\_id, 217
  - AK\_sequence\_modify, 217
  - AK\_sequence\_next\_value, 218
  - AK\_sequence\_remove, 218
  - AK\_sequence\_rename, 218
  - AK\_sequence\_test, 219
- sequence.h
  - AK\_sequence\_add, 220
  - AK\_sequence\_current\_value, 220
  - AK\_sequence\_get\_id, 220
  - AK\_sequence\_modify, 220
  - AK\_sequence\_next\_value, 221
  - AK\_sequence\_remove, 221
  - AK\_sequence\_rename, 221
  - AK\_sequence\_test, 222
- sql/cs/between.c, 228
- sql/cs/between.h, 230
- sql/cs/check\_constraint.c, 232
- sql/cs/check\_constraint.h, 234
- sql/cs/constraint\_names.c, 236
- sql/cs/constraint\_names.h, 237
- sql/cs/nnull.c, 238
- sql/cs/nnull.h, 239
- sql/cs/reference.c, 241
- sql/cs/reference.h, 244
- sql/cs/unique.c, 251
- sql/cs/unique.h, 253
- sql/drop.c, 254
- sql/drop.h, 257
- sql/function.c, 258
- sql/function.h, 262
- sql/privileges.c, 266
- sql/select.c, 274
- sql/trigger.c, 275
- sql/trigger.h, 279
- sql/view.c, 282
- struct\_add, 26
- system\_catalog
  - drop.c, 256
- table.c
  - AK\_check\_tables\_scheme, 118
  - AK\_get\_attr\_index, 118
  - AK\_get\_attr\_name, 118
  - AK\_get\_column, 119
  - AK\_get\_header, 119
  - AK\_get\_num\_records, 119
  - AK\_get\_row, 120
  - AK\_get\_table\_obj\_id, 120
  - AK\_get\_tuple, 120
  - AK\_num\_attr, 121
  - AK\_op\_rename\_test, 121
  - AK\_print\_row, 121
  - AK\_print\_row\_spacer, 122
  - AK\_print\_row\_spacer\_to\_file, 122
  - AK\_print\_row\_to\_file, 122
  - AK\_print\_table, 123
  - AK\_print\_table\_to\_file, 123
  - AK\_rename, 123
  - AK\_table\_empty, 124
  - AK\_table\_exist, 124
  - AK\_table\_test, 124
  - AK\_tuple\_to\_string, 124
  - get\_row\_attr\_data, 125
- table.h
  - AK\_check\_tables\_scheme, 127
  - AK\_get\_attr\_index, 127
  - AK\_get\_attr\_name, 127
  - AK\_get\_column, 128
  - AK\_get\_header, 128
  - AK\_get\_num\_records, 128
  - AK\_get\_row, 129
  - AK\_get\_table\_obj\_id, 129
  - AK\_get\_tuple, 129
  - AK\_num\_attr, 130
  - AK\_op\_rename\_test, 130
  - AK\_print\_row, 130
  - AK\_print\_row\_spacer, 131
  - AK\_print\_row\_spacer\_to\_file, 131
  - AK\_print\_row\_to\_file, 131
  - AK\_print\_table, 132
  - AK\_print\_table\_to\_file, 132
  - AK\_rename, 132
  - AK\_table\_empty, 133
  - AK\_table\_test, 133
  - AK\_tuple\_to\_string, 133
  - get\_row\_attr\_data, 133
- table\_addresses, 27
- test.c
  - AK\_create\_test\_tables, 135
  - AK\_get\_table\_attribute\_types, 135
  - create\_header\_test, 135
  - get\_column\_test, 135
  - get\_row\_test, 136
  - insert\_data\_test, 136
  - selection\_test, 136
- test.h
  - AK\_create\_test\_tables, 137
  - AK\_get\_table\_attribute\_types, 138
  - create\_header\_test, 138
  - get\_column\_test, 138
  - get\_row\_test, 138
  - insert\_data\_test, 139
  - selection\_test, 139
- theta\_join.c
  - AK\_check\_constraints, 222
  - AK\_create\_theta\_join\_header, 223

- AK\_op\_theta\_join\_test, 223
- AK\_theta\_join, 223
- theta\_join.h
  - AK\_check\_constraints, 224
  - AK\_create\_theta\_join\_header, 225
  - AK\_op\_theta\_join\_test, 225
  - AK\_theta\_join, 225
- threadContainer, 27
- trans/transaction.c, 285
- trans/transaction.h, 296
- transaction.c
  - AK\_acquire\_lock, 287
  - AK\_add\_hash\_entry\_list, 287
  - AK\_add\_lock, 288
  - AK\_all\_transactions\_finished, 288
  - AK\_create\_lock, 288
  - AK\_create\_new\_transaction\_thread, 288
  - AK\_delete\_hash\_entry\_list, 289
  - AK\_delete\_lock\_entry\_list, 289
  - AK\_execute\_commands, 289
  - AK\_execute\_transaction, 290
  - AK\_get\_memory\_blocks, 290
  - AK\_handle\_observable\_transaction\_action, 290
  - AK\_init\_observable\_transaction, 290
  - AK\_init\_observer\_lock, 291
  - AK\_isLock\_waiting, 291
  - AK\_lock\_released, 291
  - AK\_memory\_block\_hash, 291
  - AK\_on\_all\_transactions\_end, 292
  - AK\_on\_lock\_release, 292
  - AK\_on\_observable\_notify, 292
  - AK\_on\_transaction\_end, 292
  - AK\_release\_locks, 293
  - AK\_remove\_transaction\_thread, 293
  - AK\_search\_empty\_link\_for\_hook, 293
  - AK\_search\_existing\_link\_for\_hook, 293
  - AK\_search\_lock\_entry\_list\_by\_key, 294
  - AK\_transaction\_finished, 294
  - AK\_transaction\_manager, 294
  - AK\_transaction\_register\_observer, 295
  - AK\_transaction\_unregister\_observer, 295
  - handle\_transaction\_notify, 295
- transaction.h
  - AK\_acquire\_lock, 298
  - AK\_add\_hash\_entry\_list, 300
  - AK\_add\_lock, 300
  - AK\_all\_transactions\_finished, 300
  - AK\_create\_lock, 300
  - AK\_create\_new\_transaction\_thread, 301
  - AK\_delete\_hash\_entry\_list, 301
  - AK\_delete\_lock\_entry\_list, 301
  - AK\_execute\_commands, 302
  - AK\_execute\_transaction, 302
  - AK\_get\_memory\_blocks, 302
  - AK\_handle\_observable\_transaction\_action, 303
  - AK\_init\_observable\_transaction, 303
  - AK\_init\_observer\_lock, 303
  - AK\_isLock\_waiting, 303
  - AK\_lock\_released, 304
  - AK\_memory\_block\_hash, 304
  - AK\_on\_all\_transactions\_end, 304
  - AK\_on\_lock\_release, 304
  - AK\_on\_observable\_notify, 304
  - AK\_on\_transaction\_end, 305
  - AK\_release\_locks, 305
  - AK\_remove\_transaction\_thread, 305
  - AK\_search\_empty\_link\_for\_hook, 305
  - AK\_search\_existing\_link\_for\_hook, 306
  - AK\_search\_lock\_entry\_list\_by\_key, 306
  - AK\_transaction\_finished, 306
  - AK\_transaction\_manager, 306
  - AK\_transaction\_register\_observer, 307
  - AK\_transaction\_unregister\_observer, 307
  - handle\_transaction\_notify, 307
  - NoticeType, 298
- transaction\_list\_elem, 28
- transaction\_list\_head, 28
- transaction\_locks\_list\_elem, 29
- transactionData, 29
- trigger.c
  - AK\_trigger\_add, 276
  - AK\_trigger\_edit, 276
  - AK\_trigger\_get\_conditions, 276
  - AK\_trigger\_get\_id, 277
  - AK\_trigger\_remove\_by\_name, 277
  - AK\_trigger\_remove\_by\_obj\_id, 277
  - AK\_trigger\_rename, 278
  - AK\_trigger\_save\_conditions, 278
  - AK\_trigger\_test, 278
- trigger.h
  - AK\_trigger\_add, 279
  - AK\_trigger\_edit, 280
  - AK\_trigger\_get\_conditions, 280
  - AK\_trigger\_get\_id, 280
  - AK\_trigger\_remove\_by\_name, 281
  - AK\_trigger\_remove\_by\_obj\_id, 281
  - AK\_trigger\_rename, 281
  - AK\_trigger\_save\_conditions, 281
  - AK\_trigger\_test, 282
- union.c
  - AK\_op\_union\_test, 226
  - AK\_union, 226
- union.h
  - AK\_op\_union\_test, 227
  - AK\_union, 227
- unique.c
  - AK\_delete\_constraint\_unique, 251
  - AK\_read\_constraint\_unique, 251
  - AK\_unique\_test, 252
  - Ak\_set\_constraint\_unique, 252
- unique.h
  - AK\_delete\_constraint\_unique, 253
  - AK\_read\_constraint\_unique, 253
  - AK\_unique\_test, 254
  - Ak\_set\_constraint\_unique, 254

## view.c

- AK\_get\_rel\_exp, [283](#)
- AK\_get\_view\_obj\_id, [283](#)
- AK\_get\_view\_query, [283](#)
- AK\_view\_add, [283](#)
- AK\_view\_change\_query, [284](#)
- AK\_view\_remove\_by\_name, [284](#)
- AK\_view\_remove\_by\_obj\_id, [284](#)
- AK\_view\_rename, [285](#)
- AK\_view\_test, [285](#)