Kalashnikov DB

0.9.3

Generated by Doxygen 1.8.17

1 Todo List 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 7

3.1 File List . . . 7

4 Class Documentation 11

4.1 _dictionary_ Struct Reference . . . 11

4.1.1 Detailed Description . . . 11

4.1.2 Member Data Documentation . . . 11

4.1.2.1 hash . . . 11

4.1.2.2 key . . . 12

4.1.2.3 size . . . 12

4.1.2.4 val . . . 12

4.2 _file_metadata Struct Reference . . . 12

4.3 _notifyDetails Struct Reference . . . 12

4.4 AK_agg_input Struct Reference . . . 13

4.4.1 Detailed Description . . . 13

4.5 AK_agg_value Struct Reference . . . 13

4.5.1 Detailed Description . . . 13

4.6 AK_block Struct Reference . . . 14

4.6.1 Detailed Description . . . 14

4.7 AK_block_activity Struct Reference . . . 14

4.7.1 Detailed Description . . . 15

4.8 AK_blocktable Struct Reference . . . 15

4.9 AK_command_recovery_struct Struct Reference . . . 15

4.9.1 Detailed Description . . . 16

4.10 AK_command_struct Struct Reference . . . 16

4.11 AK_create_table_struct Struct Reference . . . 16

4.12 AK_db_cache Struct Reference . . . 17

4.12.1 Detailed Description . . . 17

4.13 AK_debmod_state Struct Reference . . . 17

4.13.1 Detailed Description . . . 18

4.14 AK_header Struct Reference . . . 18

4.14.1 Detailed Description . . . 18

4.15 AK_mem_block Struct Reference . . . 19

4.15.1 Detailed Description . . . 19

4.16 AK_operand Struct Reference . . . 19

4.17 AK_query_mem Struct Reference . . . 20

4.17.1 Detailed Description . . . 20

4.18 AK_query_mem_dict Struct Reference . . . 20

4.18.1 Detailed Description . . . 21
4.19 AK_query_mem_lib Struct Reference . . . 21
4.19.1 Detailed Description . . . 21
4.20 AK_query_mem_result Struct Reference . . . 21
4.20.1 Detailed Description . . . 22
4.21 AK_redo_log Struct Reference . . . 22
4.21.1 Detailed Description . . . 22
4.22 AK_ref_item Struct Reference . . . 23
4.22.1 Detailed Description . . . 23
4.23 AK_results Struct Reference . . . 23
4.23.1 Detailed Description . . . 24
4.24 AK_synchronization_info Struct Reference . . . 24
4.24.1 Detailed Description . . . 24
4.25 AK_tuple_dict Struct Reference . . . 24
4.25.1 Detailed Description . . . 25
4.26 blocktable Struct Reference . . . 25
4.26.1 Detailed Description . . . 25
4.27 btree_node Struct Reference . . . 25
4.28 bucket_elem Struct Reference . . . 26
4.28.1 Detailed Description . . . 26
4.29 cost_eval_t Struct Reference . . . 26
4.29.1 Detailed Description . . . 27
4.30 DEBUG_LEVEL Struct Reference . . . 27
4.30.1 Detailed Description . . . 27
4.31 DEBUG_TYPE Struct Reference . . . 27
4.31.1 Detailed Description . . . 28
4.32 drop_arguments Struct Reference . . . 28
4.33 hash_bucket Struct Reference . . . 28
4.33.1 Detailed Description . . . 29
4.34 hash_info Struct Reference . . . 29
4.34.1 Detailed Description . . . 29
4.35 intersect_attr Struct Reference . . . 29
4.35.1 Detailed Description . . . 30
4.36 list_node Struct Reference . . . 30
4.36.1 Detailed Description . . . 30
4.37 list_structure_ad Struct Reference . . . 31
4.38 list_structure_add Struct Reference . . . 31
4.38.1 Detailed Description . . . 31
4.39 main_bucket Struct Reference . . . 31
4.39.1 Detailed Description . . . 32
4.40 memoryAddresses Struct Reference . . . 32
4.40.1 Detailed Description . . . 32

4.41 Observable Struct Reference . . . 32
4.41.1 Detailed Description . . . 33
4.42 observable_transaction Struct Reference . . . 33
4.42.1 Detailed Description . . . 33
4.43 observable_transaction_struct Struct Reference . . . 33
4.44 Observer Struct Reference . . . 34
4.44.1 Detailed Description . . . 34
4.45 observer_lock Struct Reference . . . 34
4.45.1 Detailed Description . . . 35
4.46 root_info Struct Reference . . . 35
4.47 search_params Struct Reference . . . 35
4.47.1 Detailed Description . . . 36
4.48 search_result Struct Reference . . . 36
4.48.1 Detailed Description . . . 36
4.49 Stack Struct Reference . . . 37
4.49.1 Detailed Description . . . 37
4.50 struct_add Struct Reference . . . 37
4.50.1 Detailed Description . . . 37
4.51 Succesor Struct Reference . . . 38
4.51.1 Detailed Description . . . 38
4.52 table_addresses Struct Reference . . . 38
4.52.1 Detailed Description . . . 38
4.53 TestResult Struct Reference . . . 39
4.53.1 Detailed Description . . . 39
4.54 threadContainer Struct Reference . . . 39
4.54.1 Detailed Description . . . 40
4.55 transaction_list_elem Struct Reference . . . 40
4.55.1 Detailed Description . . . 40
4.56 transaction_list_head Struct Reference . . . 41
4.56.1 Detailed Description . . . 41
4.57 transaction_locks_list_elem Struct Reference . . . 41
4.57.1 Detailed Description . . . 41
4.58 transactionData Struct Reference . . . 42
4.58.1 Detailed Description . . . 42
4.59 TypeObservable Struct Reference . . . 42
4.60 TypeObserver Struct Reference . . . 42
4.61 Vertex Struct Reference . . . 43
4.61.1 Detailed Description . . . 43

5 File Documentation **45**

5.1 auxi/auxiliary.c File Reference . . . 45
5.2 auxi/auxiliary.h File Reference . . . 45

5.2.1 Detailed Description . . . 47
5.2.2 Function Documentation . . . 48
5.2.2.1 AK_add_succesor() . . . 48
5.2.2.2 AK_add_vertex() . . . 48
5.2.2.3 AK_chars_num_from_number() . . . 49
5.2.2.4 AK_convert_type() . . . 49
5.2.2.5 AK_Delete_L3() . . . 50
5.2.2.6 AK_DeleteAll_L3() . . . 50
5.2.2.7 AK_destroy_critical_section() . . . 51
5.2.2.8 AK_End_L2() . . . 51
5.2.2.9 AK_enter_critical_section() . . . 51
5.2.2.10 AK_First_L2() . . . 52
5.2.2.11 AK_get_array_perms() . . . 52
5.2.2.12 AK_GetNth_L2() . . . 53
5.2.2.13 AK_init_critical_section() . . . 55
5.2.2.14 AK_Init_L3() . . . 55
5.2.2.15 AK_InsertAfter_L2() . . . 55
5.2.2.16 AK_InsertAtBegin_L3() . . . 56
5.2.2.17 AK_InsertAtEnd_L3() . . . 57
5.2.2.18 AK_InsertBefore_L2() . . . 57
5.2.2.19 AK_IsEmpty_L2() . . . 58
5.2.2.20 AK_leave_critical_section() . . . 58
5.2.2.21 AK_Next_L2() . . . 58
5.2.2.22 AK_pop_from_stack() . . . 59
5.2.2.23 AK_Previous_L2() . . . 59
5.2.2.24 AK_push_to_stack() . . . 60
5.2.2.25 AK_Retrieve_L2() . . . 60
5.2.2.26 AK_search_empty_link() . . . 60
5.2.2.27 AK_search_empty_stack_link() . . . 61
5.2.2.28 AK_search_in_stack() . . . 61
5.2.2.29 AK_search_vertex() . . . 62
5.2.2.30 AK_Size_L2() . . . 62
5.2.2.31 AK_strcmp() . . . 63
5.2.2.32 AK_tarjan() . . . 63
5.2.2.33 AK_type_size() . . . 63
5.2.3 Variable Documentation . . . 64
5.2.3.1 testMode . . . 64
5.3 auxi/constants.h File Reference . . . 64
5.3.1 Detailed Description . . . 68
5.3.2 Macro Definition Documentation . . . 68
5.3.2.1 AK_CONSTRAINTS_DEFAULT . . . 68
5.3.2.2 AK_CONSTRAINTS_FOREIGN_KEY . . . 69

5.3.2.3 AK_CONSTRAINTS_INDEX . . . 69
5.3.2.4 AK_CONSTRAINTS_PRIMARY_KEY . . . 69
5.4 auxi/debug.c File Reference . . . 69
5.4.1 Detailed Description . . . 69
5.4.2 Function Documentation . . . 70
5.4.2.1 AK_dbg_messg() . . . 70
5.5 auxi/debug.h File Reference . . . 70
5.5.1 Detailed Description . . . 71
5.5.2 Macro Definition Documentation . . . 71
5.5.2.1 DEBUG_ALL . . . 71
5.5.3 Function Documentation . . . 71
5.5.3.1 AK_dbg_messg() . . . 71
5.6 auxi/dictionary.c File Reference . . . 72
5.6.1 Detailed Description . . . 73
5.6.2 Macro Definition Documentation . . . 73
5.6.2.1 DICT_INVALID_KEY . . . 73
5.6.2.2 DICTMINSZ . . . 73
5.6.2.3 MAXVALSZ . . . 73
5.6.3 Function Documentation . . . 73
5.6.3.1 dictionary_del() . . . 73
5.6.3.2 dictionary_dump() . . . 74
5.6.3.3 dictionary_get() . . . 74
5.6.3.4 dictionary_hash() . . . 75
5.6.3.5 dictionary_new() . . . 75
5.6.3.6 dictionary_set() . . . 75
5.6.3.7 dictionary_unset() . . . 76
5.7 auxi/dictionary.h File Reference . . . 76
5.7.1 Detailed Description . . . 77
5.7.2 Typedef Documentation . . . 77
5.7.2.1 dictionary . . . 77
5.7.3 Function Documentation . . . 77
5.7.3.1 dictionary_del() . . . 77
5.7.3.2 dictionary_dump() . . . 78
5.7.3.3 dictionary_get() . . . 78
5.7.3.4 dictionary_hash() . . . 79
5.7.3.5 dictionary_new() . . . 79
5.7.3.6 dictionary_set() . . . 79
5.7.3.7 dictionary_unset() . . . 80
5.8 auxi/iniparser.c File Reference . . . 80
5.8.1 Detailed Description . . . 82
5.8.2 Typedef Documentation . . . 82
5.8.2.1 line_status . . . 82

5.8.3 Enumeration Type Documentation . . . 82
5.8.3.1 _line_status_ . . . 82
5.8.4 Function Documentation . . . 82
5.8.4.1 iniparser_AK_freedict() . . . 82
5.8.4.2 iniparser_dump() . . . 83
5.8.4.3 iniparser_dump_ini() . . . 83
5.8.4.4 iniparser_dumpsection_ini() . . . 84
5.8.4.5 iniparser_find_entry() . . . 84
5.8.4.6 iniparser_getboolean() . . . 84
5.8.4.7 iniparser_getdouble() . . . 85
5.8.4.8 iniparser_getint() . . . 86
5.8.4.9 iniparser_getnsec() . . . 86
5.8.4.10 iniparser_getseckeys() . . . 87
5.8.4.11 iniparser_getsecname() . . . 87
5.8.4.12 iniparser_getsecnkeys() . . . 88
5.8.4.13 iniparser_getstring() . . . 88
5.8.4.14 iniparser_load() . . . 89
5.8.4.15 iniparser_set() . . . 89
5.8.4.16 iniparser_unset() . . . 89
5.9 auxi/iniparser.h File Reference . . . 90
5.9.1 Detailed Description . . . 91
5.9.2 Function Documentation . . . 91
5.9.2.1 iniparser_AK_freedict() . . . 91
5.9.2.2 iniparser_dump() . . . 91
5.9.2.3 iniparser_dump_ini() . . . 92
5.9.2.4 iniparser_dumpsection_ini() . . . 92
5.9.2.5 iniparser_find_entry() . . . 93
5.9.2.6 iniparser_getboolean() . . . 93
5.9.2.7 iniparser_getdouble() . . . 94
5.9.2.8 iniparser_getint() . . . 94
5.9.2.9 iniparser_getnsec() . . . 95
5.9.2.10 iniparser_getseckeys() . . . 96
5.9.2.11 iniparser_getsecname() . . . 96
5.9.2.12 iniparser_getsecnkeys() . . . 97
5.9.2.13 iniparser_getstring() . . . 97
5.9.2.14 iniparser_load() . . . 98
5.9.2.15 iniparser_set() . . . 98
5.9.2.16 iniparser_unset() . . . 98
5.10 auxi/mempro.c File Reference . . . 99
5.10.1 Detailed Description . . . 100
5.10.2 Function Documentation . . . 100
5.10.2.1 AK_calloc() . . . 100

5.10.2.2 AK_check_for_writes() . . . 101
5.10.2.3 AK_debmod_calloc() . . . 101
5.10.2.4 AK_debmod_d() . . . 102
5.10.2.5 AK_debmod_die() . . . 102
5.10.2.6 AK_debmod_dv() . . . 103
5.10.2.7 AK_debmod_enter_critical_sec() . . . 103
5.10.2.8 AK_debmod_free() . . . 103
5.10.2.9 AK_debmod_fstack_pop() . . . 104
5.10.2.10 AK_debmod_fstack_push() . . . 104
5.10.2.11 AK_debmod_func_add() . . . 105
5.10.2.12 AK_debmod_func_get_name() . . . 105
5.10.2.13 AK_debmod_func_id() . . . 106
5.10.2.14 AK_debmod_function_current() . . . 106
5.10.2.15 AK_debmod_function_epilogue() . . . 107
5.10.2.16 AK_debmod_function_prologue() . . . 107
5.10.2.17 AK_debmod_init() . . . 108
5.10.2.18 AK_debmod_leave_critical_sec() . . . 108
5.10.2.19 AK_debmod_log_memory_alloc() . . . 109
5.10.2.20 AK_debmod_print_function_use() . . . 109
5.10.2.21 AK_fread() . . . 109
5.10.2.22 AK_free() . . . 110
5.10.2.23 AK_fwrite() . . . 110
5.10.2.24 AK_malloc() . . . 111
5.10.2.25 AK_mempro_test() . . . 111
5.10.2.26 AK_print_active_functions() . . . 111
5.10.2.27 AK_print_function_use() . . . 112
5.10.2.28 AK_print_function_uses() . . . 112
5.10.2.29 AK_realloc() . . . 112
5.10.2.30 AK_write_protect() . . . 113
5.10.2.31 AK_write_unprotect() . . . 113
5.11 auxi/mempro.h File Reference . . . 114
5.11.1 Detailed Description . . . 116
5.11.2 Function Documentation . . . 116
5.11.2.1 AK_calloc() . . . 116
5.11.2.2 AK_check_for_writes() . . . 117
5.11.2.3 AK_debmod_calloc() . . . 117
5.11.2.4 AK_debmod_d() . . . 117
5.11.2.5 AK_debmod_die() . . . 118
5.11.2.6 AK_debmod_dv() . . . 118
5.11.2.7 AK_debmod_enter_critical_sec() . . . 119
5.11.2.8 AK_debmod_free() . . . 119
5.11.2.9 AK_debmod_fstack_pop() . . . 120

5.11.2.10 AK_debmod_fstack_push() . . . 120
5.11.2.11 AK_debmod_func_add() . . . 120
5.11.2.12 AK_debmod_func_get_name() . . . 121
5.11.2.13 AK_debmod_func_id() . . . 121
5.11.2.14 AK_debmod_function_current() . . . 122
5.11.2.15 AK_debmod_function_epilogue() . . . 122
5.11.2.16 AK_debmod_function_prologue() . . . 123
5.11.2.17 AK_debmod_init() . . . 123
5.11.2.18 AK_debmod_leave_critical_sec() . . . 124
5.11.2.19 AK_debmod_log_memory_alloc() . . . 124
5.11.2.20 AK_debmod_print_function_use() . . . 124
5.11.2.21 AK_free() . . . 125
5.11.2.22 AK_malloc() . . . 125
5.11.2.23 AK_mempro_test() . . . 126
5.11.2.24 AK_print_active_functions() . . . 126
5.11.2.25 AK_print_function_use() . . . 126
5.11.2.26 AK_print_function_uses() . . . 127
5.11.2.27 AK_realloc() . . . 127
5.11.2.28 AK_write_protect() . . . 128
5.11.2.29 AK_write_unprotect() . . . 128
5.12 auxi/observable.c File Reference . . . 128
5.12.1 Detailed Description . . . 129
5.12.2 Function Documentation . . . 129
5.12.2.1 AK_init_observable() . . . 130
5.12.2.2 AK_init_observer() . . . 130
5.12.2.3 AK_observable_test() . . . 130
5.13 auxi/observable.h File Reference . . . 131
5.13.1 Detailed Description . . . 131
5.13.2 Function Documentation . . . 131
5.13.2.1 AK_init_observable() . . . 132
5.13.2.2 AK_init_observer() . . . 132
5.13.2.3 AK_observable_test() . . . 132
5.14 auxi/test.c File Reference . . . 132
5.14.1 Detailed Description . . . 133
5.14.2 Function Documentation . . . 133
5.14.2.1 TEST_output_results() . . . 133
5.14.2.2 TEST_result() . . . 133
5.15 file/test.c File Reference . . . 134
5.15.1 Detailed Description . . . 134
5.15.2 Function Documentation . . . 134
5.15.2.1 AK_create_test_tables() . . . 134
5.15.2.2 AK_get_table_atribute_types() . . . 135

5.15.2.3 create_header_test() . . . 135
5.15.2.4 get_column_test() . . . 135
5.15.2.5 get_row_test() . . . 136
5.15.2.6 insert_data_test() . . . 136
5.15.2.7 selection_test() . . . 137
5.16 file/test.h File Reference . . . 138
5.16.1 Detailed Description . . . 138
5.16.2 Function Documentation . . . 138
5.16.2.1 AK_create_test_tables() . . . 138
5.16.2.2 AK_get_table_atribute_types() . . . 139
5.16.2.3 create_header_test() . . . 139
5.16.2.4 get_column_test() . . . 139
5.16.2.5 get_row_test() . . . 140
5.16.2.6 insert_data_test() . . . 140
5.16.2.7 selection_test() . . . 141
5.17 dm/dbman.c File Reference . . . 142
5.17.1 Detailed Description . . . 144
5.17.2 Function Documentation . . . 144
5.17.2.1 AK_allocate_block_activity_modes() . . . 144
5.17.2.2 AK_allocate_blocks() . . . 144
5.17.2.3 AK_allocationtable_dump() . . . 145
5.17.2.4 AK_blocktable_dump() . . . 145
5.17.2.5 AK_blocktable_flush() . . . 145
5.17.2.6 AK_blocktable_get() . . . 146
5.17.2.7 AK_copy_header() . . . 146
5.17.2.8 AK_create_header() . . . 146
5.17.2.9 AK_delete_block() . . . 147
5.17.2.10 AK_delete_extent() . . . 147
5.17.2.11 AK_delete_segment() . . . 148
5.17.2.12 AK_get_allocation_set() . . . 148
5.17.2.13 AK_get_extent() . . . 149
5.17.2.14 AK_increase_extent() . . . 150
5.17.2.15 AK_init_allocation_table() . . . 150
5.17.2.16 AK_init_block() . . . 151
5.17.2.17 AK_init_db_file() . . . 151
5.17.2.18 AK_init_disk_manager() . . . 151
5.17.2.19 AK_init_system_catalog() . . . 152
5.17.2.20 AK_init_system_tables_catalog() . . . 152
5.17.2.21 AK_insert_entry() . . . 153
5.17.2.22 AK_memset_int() . . . 154
5.17.2.23 AK_new_extent() . . . 155
5.17.2.24 AK_new_segment() . . . 155

5.17.2.25 AK_print_block() . . . 156
5.17.2.26 AK_read_block() . . . 156
5.17.2.27 AK_read_block_for_testing() . . . 157
5.17.2.28 AK_register_system_tables() . . . 157
5.17.2.29 AK_thread_safe_block_access_test() . . . 158
5.17.2.30 AK_write_block() . . . 158
5.17.2.31 AK_write_block_for_testing() . . . 159
5.17.2.32 fsize() . . . 159
5.18 dm/dbman.h File Reference . . . 159
5.18.1 Detailed Description . . . 163
5.18.2 Macro Definition Documentation . . . 163
5.18.2.1 AK_ALLOCATION_TABLE_SIZE . . . 163
5.18.2.2 CHAR_IN_LINE . . . 163
5.18.2.3 MAX_BLOCK_INIT_NUM . . . 163
5.18.3 Enumeration Type Documentation . . . 163
5.18.3.1 AK_allocation_set_mode . . . 164
5.18.4 Function Documentation . . . 164
5.18.4.1 AK_allocate_blocks() . . . 164
5.18.4.2 AK_allocationtable_dump() . . . 164
5.18.4.3 AK_blocktable_dump() . . . 165
5.18.4.4 AK_blocktable_flush() . . . 165
5.18.4.5 AK_blocktable_get() . . . 165
5.18.4.6 AK_copy_header() . . . 166
5.18.4.7 AK_create_header() . . . 166
5.18.4.8 AK_delete_block() . . . 167
5.18.4.9 AK_delete_extent() . . . 167
5.18.4.10 AK_delete_segment() . . . 168
5.18.4.11 AK_get_allocation_set() . . . 168
5.18.4.12 AK_get_extent() . . . 169
5.18.4.13 AK_increase_extent() . . . 169
5.18.4.14 AK_init_allocation_table() . . . 170
5.18.4.15 AK_init_block() . . . 170
5.18.4.16 AK_init_db_file() . . . 171
5.18.4.17 AK_init_disk_manager() . . . 171
5.18.4.18 AK_init_system_catalog() . . . 171
5.18.4.19 AK_init_system_tables_catalog() . . . 172
5.18.4.20 AK_insert_entry() . . . 173
5.18.4.21 AK_memset_int() . . . 174
5.18.4.22 AK_new_extent() . . . 174
5.18.4.23 AK_new_segment() . . . 175
5.18.4.24 AK_print_block() . . . 175
5.18.4.25 AK_read_block() . . . 176

5.18.4.26 AK_read_block_for_testing() . . . 176
5.18.4.27 AK_register_system_tables() . . . 176
5.18.4.28 AK_thread_safe_block_access_test() . . . 177
5.18.4.29 AK_write_block() . . . 178
5.18.4.30 AK_write_block_for_testing() . . . 178
5.18.4.31 fsize() . . . 178
5.18.5 Variable Documentation . . . 179
5.18.5.1 AK_allocationbit . . . 179
5.18.5.2 db . . . 179
5.18.5.3 db_file_size . . . 179
5.19 file/blobs.c File Reference . . . 179
5.19.1 Detailed Description . . . 180
5.19.2 Function Documentation . . . 180
5.19.2.1 AK_check_folder_blobs() . . . 180
5.19.2.2 AK_concat() . . . 181
5.19.2.3 AK_folder_exists() . . . 181
5.19.2.4 AK_GUID() . . . 181
5.19.2.5 AK_lo_export() . . . 182
5.19.2.6 AK_lo_import() . . . 182
5.19.2.7 AK_lo_test() . . . 182
5.19.2.8 AK_lo_unlink() . . . 183
5.19.2.9 AK_mkdir() . . . 183
5.19.2.10 AK_split_path_file() . . . 183
5.20 file/blobs.h File Reference . . . 184
5.20.1 Detailed Description . . . 184
5.20.2 Function Documentation . . . 185
5.20.2.1 AK_check_folder_blobs() . . . 185
5.20.2.2 AK_concat() . . . 185
5.20.2.3 AK_folder_exists() . . . 185
5.20.2.4 AK_GUID() . . . 186
5.20.2.5 AK_lo_export() . . . 186
5.20.2.6 AK_lo_import() . . . 186
5.20.2.7 AK_lo_test() . . . 187
5.20.2.8 AK_lo_unlink() . . . 187
5.20.2.9 AK_mkdir() . . . 187
5.20.2.10 AK_split_path_file() . . . 188
5.21 file/fileio.c File Reference . . . 188
5.21.1 Detailed Description . . . 189
5.21.2 Function Documentation . . . 189
5.21.2.1 AK_delete_row() . . . 189
5.21.2.2 AK_delete_row_by_id() . . . 189
5.21.2.3 AK_delete_row_from_block() . . . 189

5.21.2.4 AK_delete_update_segment() . . . 190
5.21.2.5 AK_Insert_New_Element() . . . 190
5.21.2.6 AK_Insert_New_Element_For_Update() . . . 191
5.21.2.7 AK_insert_row() . . . 192
5.21.2.8 AK_insert_row_to_block() . . . 192
5.21.2.9 AK_Update_Existing_Element() . . . 193
5.21.2.10 AK_update_row() . . . 193
5.21.2.11 AK_update_row_from_block() . . . 194
5.22 file/fileio.h File Reference . . . 194
5.22.1 Detailed Description . . . 195
5.22.2 Function Documentation . . . 195
5.22.2.1 AK_delete_row() . . . 195
5.22.2.2 AK_delete_row_by_id() . . . 196
5.22.2.3 AK_delete_row_from_block() . . . 196
5.22.2.4 AK_delete_update_segment() . . . 196
5.22.2.5 AK_Insert_New_Element() . . . 197
5.22.2.6 AK_Insert_New_Element_For_Update() . . . 197
5.22.2.7 AK_insert_row() . . . 198
5.22.2.8 AK_insert_row_to_block() . . . 199
5.22.2.9 AK_update_row() . . . 199
5.22.2.10 AK_update_row_from_block() . . . 200
5.23 file/files.c File Reference . . . 200
5.23.1 Detailed Description . . . 200
5.23.2 Function Documentation . . . 201
5.23.2.1 AK_files_test() . . . 201
5.23.2.2 AK_initialize_new_index_segment() . . . 201
5.23.2.3 AK_initialize_new_segment() . . . 202
5.24 file/files.h File Reference . . . 202
5.24.1 Detailed Description . . . 202
5.24.2 Function Documentation . . . 203
5.24.2.1 AK_files_test() . . . 203
5.24.2.2 AK_initialize_new_index_segment() . . . 203
5.24.2.3 AK_initialize_new_segment() . . . 204
5.25 file/filesearch.c File Reference . . . 204
5.25.1 Detailed Description . . . 204
5.25.2 Function Documentation . . . 205
5.25.2.1 AK_deallocate_search_result() . . . 205
5.25.2.2 AK_filesearch_test() . . . 205
5.25.2.3 AK_search_unsorted() . . . 206
5.26 file/filesearch.h File Reference . . . 206
5.26.1 Detailed Description . . . 207
5.26.2 Function Documentation . . . 207

5.26.2.1 AK_deallocate_search_result() . . . 207
5.26.2.2 AK_filesearch_test() . . . 208
5.26.2.3 AK_search_unsorted() . . . 208
5.27 file/filesort.h File Reference . . . 209
5.27.1 Detailed Description . . . 209
5.27.2 Function Documentation . . . 209
5.27.2.1 AK_block_sort() . . . 210
5.27.2.2 AK_get_header_number() . . . 210
5.27.2.3 AK_get_num_of_tuples() . . . 211
5.27.2.4 AK_get_total_headers() . . . 211
5.27.2.5 AK_reset_block() . . . 211
5.27.2.6 AK_sort_segment() . . . 212
5.28 file/id.c File Reference . . . 212
5.28.1 Detailed Description . . . 213
5.28.2 Function Documentation . . . 213
5.28.2.1 AK_get_id() . . . 213
5.28.2.2 AK_get_table_id() . . . 213
5.28.2.3 AK_id_test() . . . 214
5.29 file/id.h File Reference . . . 214
5.29.1 Detailed Description . . . 214
5.29.2 Function Documentation . . . 214
5.29.2.1 AK_get_id() . . . 215
5.29.2.2 AK_id_test() . . . 215
5.30 file/idx/bitmap.c File Reference . . . 215
5.30.1 Detailed Description . . . 216
5.30.2 Function Documentation . . . 216
5.30.2.1 AK_add_to_bitmap_index() . . . 216
5.30.2.2 AK_bitmap_test() . . . 217
5.30.2.3 AK_create_Index() . . . 218
5.30.2.4 AK_create_Index_Table() . . . 218
5.30.2.5 AK_delete_bitmap_index() . . . 219
5.30.2.6 AK_get_attribute() . . . 219
5.30.2.7 AK_get_Attribute() . . . 220
5.30.2.8 AK_If_ExistOp() . . . 220
5.30.2.9 AK_print_Att_Test() . . . 221
5.30.2.10 AK_print_Header_Test() . . . 221
5.30.2.11 AK_update() . . . 221
5.31 file/idx/bitmap.h File Reference . . . 222
5.31.1 Detailed Description . . . 223
5.31.2 Function Documentation . . . 223
5.31.2.1 AK_add_to_bitmap_index() . . . 223
5.31.2.2 AK_bitmap_test() . . . 224

5.31.2.3 AK_create_Index() . . . 225
5.31.2.4 AK_create_Index_Table() . . . 225
5.31.2.5 AK_delete_bitmap_index() . . . 226
5.31.2.6 AK_get_attribute() . . . 226
5.31.2.7 AK_get_Attribute() . . . 227
5.31.2.8 AK_If_ExistOp() . . . 227
5.31.2.9 AK_print_Att_Test() . . . 228
5.31.2.10 AK_print_Header_Test() . . . 228
5.31.2.11 AK_update() . . . 228
5.31.2.12 AK_write_block() . . . 229
5.32 file/idx/btree.c File Reference . . . 230
5.32.1 Detailed Description . . . 230
5.32.2 Function Documentation . . . 230
5.32.2.1 AK_btree_create() . . . 230
5.32.2.2 AK_btree_search_delete() . . . 231
5.33 file/idx/btree.h File Reference . . . 231
5.33.1 Detailed Description . . . 232
5.33.2 Function Documentation . . . 232
5.33.2.1 AK_btree_create() . . . 232
5.33.2.2 AK_btree_search_delete() . . . 232
5.34 file/idx/hash.c File Reference . . . 233
5.34.1 Detailed Description . . . 233
5.34.2 Function Documentation . . . 234
5.34.2.1 AK_change_hash_info() . . . 234
5.34.2.2 AK_create_hash_index() . . . 234
5.34.2.3 AK_delete_in_hash_index() . . . 235
5.34.2.4 AK_elem_hash_value() . . . 235
5.34.2.5 AK_find_delete_in_hash_index() . . . 236
5.34.2.6 AK_find_in_hash_index() . . . 236
5.34.2.7 AK_get_hash_info() . . . 237
5.34.2.8 AK_get_nth_main_bucket_add() . . . 237
5.34.2.9 AK_hash_test() . . . 237
5.34.2.10 AK_insert_bucket_to_block() . . . 238
5.34.2.11 AK_insert_in_hash_index() . . . 238
5.34.2.12 AK_update_bucket_in_block() . . . 239
5.35 file/idx/hash.h File Reference . . . 239
5.35.1 Detailed Description . . . 240
5.35.2 Function Documentation . . . 240
5.35.2.1 AK_change_hash_info() . . . 241
5.35.2.2 AK_create_hash_index() . . . 241
5.35.2.3 AK_delete_in_hash_index() . . . 242
5.35.2.4 AK_elem_hash_value() . . . 242

5.35.2.5 AK_find_delete_in_hash_index() . . . 242
5.35.2.6 AK_find_in_hash_index() . . . 243
5.35.2.7 AK_get_hash_info() . . . 243
5.35.2.8 AK_get_nth_main_bucket_add() . . . 244
5.35.2.9 AK_hash_test() . . . 244
5.35.2.10 AK_insert_bucket_to_block() . . . 245
5.35.2.11 AK_insert_in_hash_index() . . . 245
5.35.2.12 AK_update_bucket_in_block() . . . 246
5.36 file/idx/index.c File Reference . . . 246
5.36.1 Detailed Description . . . 247
5.36.2 Function Documentation . . . 247
5.36.2.1 AK_Delete_All_elementsAd() . . . 247
5.36.2.2 AK_Delete_elementAd() . . . 248
5.36.2.3 AK_Get_First_elementAd() . . . 248
5.36.2.4 AK_get_index_header() . . . 248
5.36.2.5 AK_get_index_num_records() . . . 249
5.36.2.6 AK_get_index_tuple() . . . 250
5.36.2.7 AK_Get_Last_elementAd() . . . 250
5.36.2.8 AK_Get_Next_elementAd() . . . 250
5.36.2.9 AK_Get_Position_Of_elementAd() . . . 251
5.36.2.10 AK_Get_Previous_elementAd() . . . 251
5.36.2.11 AK_index_table_exist() . . . 252
5.36.2.12 AK_index_test() . . . 252
5.36.2.13 AK_InitializelistAd() . . . 253
5.36.2.14 AK_Insert_NewelementAd() . . . 253
5.36.2.15 AK_num_index_attr() . . . 254
5.36.2.16 AK_print_index_table() . . . 254
5.37 file/idx/index.h File Reference . . . 254
5.37.1 Detailed Description . . . 255
5.37.2 Function Documentation . . . 256
5.37.2.1 AK_Delete_All_elementsAd() . . . 256
5.37.2.2 AK_Delete_elementAd() . . . 256
5.37.2.3 AK_Get_First_elementAd() . . . 257
5.37.2.4 AK_get_index_num_records() . . . 257
5.37.2.5 AK_get_index_tuple() . . . 258
5.37.2.6 AK_Get_Last_elementAd() . . . 258
5.37.2.7 AK_Get_Next_elementAd() . . . 258
5.37.2.8 AK_Get_Position_Of_elementAd() . . . 259
5.37.2.9 AK_Get_Previous_elementAd() . . . 259
5.37.2.10 AK_index_table_exist() . . . 260
5.37.2.11 AK_index_test() . . . 260
5.37.2.12 AK_InitializelistAd() . . . 261

5.37.2.13 AK_Insert_NewelementAd() . . . 261
5.37.2.14 AK_num_index_attr() . . . 262
5.37.2.15 AK_print_index_table() . . . 262
5.38 file/sequence.c File Reference . . . 262
5.38.1 Detailed Description . . . 263
5.38.2 Function Documentation . . . 263
5.38.2.1 AK_sequence_add() . . . 263
5.38.2.2 AK_sequence_current_value() . . . 264
5.38.2.3 AK_sequence_get_id() . . . 264
5.38.2.4 AK_sequence_modify() . . . 265
5.38.2.5 AK_sequence_next_value() . . . 265
5.38.2.6 AK_sequence_remove() . . . 266
5.38.2.7 AK_sequence_rename() . . . 266
5.38.2.8 AK_sequence_test() . . . 267
5.39 file/sequence.h File Reference . . . 267
5.39.1 Detailed Description . . . 267
5.39.2 Function Documentation . . . 268
5.39.2.1 AK_sequence_add() . . . 268
5.39.2.2 AK_sequence_current_value() . . . 268
5.39.2.3 AK_sequence_get_id() . . . 269
5.39.2.4 AK_sequence_modify() . . . 269
5.39.2.5 AK_sequence_next_value() . . . 270
5.39.2.6 AK_sequence_remove() . . . 270
5.39.2.7 AK_sequence_rename() . . . 271
5.39.2.8 AK_sequence_test() . . . 271
5.40 file/table.c File Reference . . . 272
5.40.1 Detailed Description . . . 273
5.40.2 Function Documentation . . . 273
5.40.2.1 AK_check_tables_scheme() . . . 273
5.40.2.2 AK_create_table() . . . 273
5.40.2.3 AK_get_attr_index() . . . 274
5.40.2.4 AK_get_attr_name() . . . 274
5.40.2.5 AK_get_column() . . . 275
5.40.2.6 AK_get_header() . . . 275
5.40.2.7 AK_get_num_records() . . . 277
5.40.2.8 AK_get_row() . . . 277
5.40.2.9 AK_get_table_obj_id() . . . 278
5.40.2.10 AK_get_tuple() . . . 278
5.40.2.11 AK_num_attr() . . . 279
5.40.2.12 AK_op_rename_test() . . . 279
5.40.2.13 AK_print_row() . . . 280
5.40.2.14 AK_print_row_spacer() . . . 280

5.40.2.15 AK_print_row_spacer_to_file() . . . 280
5.40.2.16 AK_print_row_to_file() . . . 281
5.40.2.17 AK_print_table() . . . 281
5.40.2.18 AK_print_table_to_file() . . . 282
5.40.2.19 AK_rename() . . . 282
5.40.2.20 AK_table_empty() . . . 283
5.40.2.21 AK_table_exist() . . . 283
5.40.2.22 AK_table_test() . . . 284
5.40.2.23 AK_temp_create_table() . . . 284
5.40.2.24 AK_tuple_to_string() . . . 284
5.40.2.25 get_row_attr_data() . . . 285
5.41 file/table.h File Reference . . . 285
5.41.1 Detailed Description . . . 287
5.41.2 Function Documentation . . . 287
5.41.2.1 AK_check_tables_scheme() . . . 287
5.41.2.2 AK_create_table() . . . 288
5.41.2.3 AK_get_attr_index() . . . 288
5.41.2.4 AK_get_attr_name() . . . 289
5.41.2.5 AK_get_column() . . . 289
5.41.2.6 AK_get_header() . . . 289
5.41.2.7 AK_get_num_records() . . . 290
5.41.2.8 AK_get_row() . . . 291
5.41.2.9 AK_get_table_obj_id() . . . 291
5.41.2.10 AK_get_tuple() . . . 291
5.41.2.11 AK_num_attr() . . . 292
5.41.2.12 AK_op_rename_test() . . . 292
5.41.2.13 AK_print_row() . . . 293
5.41.2.14 AK_print_row_spacer() . . . 293
5.41.2.15 AK_print_row_spacer_to_file() . . . 294
5.41.2.16 AK_print_row_to_file() . . . 294
5.41.2.17 AK_print_table() . . . 295
5.41.2.18 AK_print_table_to_file() . . . 295
5.41.2.19 AK_rename() . . . 295
5.41.2.20 AK_table_empty() . . . 296
5.41.2.21 AK_table_test() . . . 296
5.41.2.22 AK_temp_create_table() . . . 297
5.41.2.23 AK_tuple_to_string() . . . 297
5.41.2.24 get_row_attr_data() . . . 298
5.42 mm/memoman.c File Reference . . . 298
5.42.1 Detailed Description . . . 299
5.42.2 Function Documentation . . . 299
5.42.2.1 AK_cache_AK_malloc() . . . 300

5.42.2.2 AK_cache_block() . . . 300
5.42.2.3 AK_cache_result() . . . 301
5.42.2.4 AK_find_AK_free_space() . . . 301
5.42.2.5 AK_find_available_result_block() . . . 301
5.42.2.6 AK_flush_cache() . . . 302
5.42.2.7 AK_generate_result_id() . . . 302
5.42.2.8 AK_get_block() . . . 302
5.42.2.9 AK_get_index_addresses() . . . 303
5.42.2.10 AK_get_index_segment_addresses() . . . 303
5.42.2.11 AK_get_segment_addresses() . . . 304
5.42.2.12 AK_get_segment_addresses_internal() . . . 304
5.42.2.13 AK_get_system_table_address() . . . 305
5.42.2.14 AK_get_table_addresses() . . . 305
5.42.2.15 AK_init_new_extent() . . . 306
5.42.2.16 AK_mem_block_modify() . . . 306
5.42.2.17 AK_memoman_init() . . . 306
5.42.2.18 AK_query_mem_AK_free() . . . 307
5.42.2.19 AK_query_mem_AK_malloc() . . . 307
5.42.2.20 AK_redo_log_AK_malloc() . . . 307
5.42.2.21 AK_refresh_cache() . . . 308
5.42.2.22 AK_release_oldest_cache_block() . . . 308
5.43 mm/memoman.h File Reference . . . 308
5.43.1 Detailed Description . . . 310
5.43.2 Function Documentation . . . 310
5.43.2.1 AK_cache_AK_malloc() . . . 310
5.43.2.2 AK_cache_block() . . . 311
5.43.2.3 AK_cache_result() . . . 311
5.43.2.4 AK_find_AK_free_space() . . . 311
5.43.2.5 AK_find_available_result_block() . . . 312
5.43.2.6 AK_flush_cache() . . . 312
5.43.2.7 AK_generate_result_id() . . . 313
5.43.2.8 AK_get_block() . . . 313
5.43.2.9 AK_get_index_addresses() . . . 314
5.43.2.10 AK_get_index_segment_addresses() . . . 314
5.43.2.11 AK_get_segment_addresses() . . . 314
5.43.2.12 AK_get_segment_addresses_internal() . . . 315
5.43.2.13 AK_get_table_addresses() . . . 316
5.43.2.14 AK_init_new_extent() . . . 316
5.43.2.15 AK_mem_block_modify() . . . 317
5.43.2.16 AK_memoman_init() . . . 317
5.43.2.17 AK_query_mem_AK_free() . . . 317
5.43.2.18 AK_query_mem_AK_malloc() . . . 318

5.43.2.19 AK_redo_log_AK_malloc() . . . 318
5.43.2.20 AK_refresh_cache() . . . 318
5.43.2.21 AK_release_oldest_cache_block() . . . 319
5.44 opti/query_optimization.c File Reference . . . 319
5.44.1 Detailed Description . . . 319
5.44.2 Function Documentation . . . 319
5.44.2.1 AK_execute_rel_eq() . . . 320
5.44.2.2 AK_print_optimized_query() . . . 320
5.44.2.3 AK_query_optimization() . . . 321
5.44.2.4 AK_query_optimization_test() . . . 321
5.45 opti/query_optimization.h File Reference . . . 321
5.45.1 Detailed Description . . . 322
5.45.2 Function Documentation . . . 322
5.45.2.1 AK_execute_rel_eq() . . . 322
5.45.2.2 AK_print_optimized_query() . . . 323
5.45.2.3 AK_query_optimization() . . . 323
5.45.2.4 AK_query_optimization_test() . . . 324
5.46 opti/rel_eq_assoc.c File Reference . . . 324
5.46.1 Detailed Description . . . 325
5.46.2 Function Documentation . . . 325
5.46.2.1 AK_compare() . . . 325
5.46.2.2 AK_print_rel_eq_assoc() . . . 325
5.46.2.3 AK_rel_eq_assoc() . . . 326
5.46.2.4 AK_rel_eq_assoc_test() . . . 326
5.47 opti/rel_eq_assoc.h File Reference . . . 326
5.47.1 Detailed Description . . . 327
5.47.2 Function Documentation . . . 327
5.47.2.1 AK_compare() . . . 327
5.47.2.2 AK_print_rel_eq_assoc() . . . 328
5.47.2.3 AK_rel_eq_assoc() . . . 328
5.47.2.4 AK_rel_eq_assoc_test() . . . 329
5.48 opti/rel_eq_comut.c File Reference . . . 329
5.48.1 Detailed Description . . . 329
5.48.2 Function Documentation . . . 329
5.48.2.1 AK_print_rel_eq_comut() . . . 329
5.48.2.2 AK_rel_eq_commute_with_theta_join() . . . 330
5.48.2.3 AK_rel_eq_comut() . . . 330
5.48.2.4 AK_rel_eq_comut_test() . . . 331
5.49 opti/rel_eq_comut.h File Reference . . . 331
5.49.1 Detailed Description . . . 331
5.49.2 Function Documentation . . . 332
5.49.2.1 AK_print_rel_eq_comut() . . . 332

5.49.2.2 AK_rel_eq_commute_with_theta_join() . . . 332
5.49.2.3 AK_rel_eq_comut() . . . 333
5.49.2.4 AK_rel_eq_comut_test() . . . 333
5.50 opti/rel_eq_projection.c File Reference . . . 333
5.50.1 Detailed Description . . . 334
5.50.2 Function Documentation . . . 334
5.50.2.1 AK_print_rel_eq_projection() . . . 334
5.50.2.2 AK_rel_eq_can_commute() . . . 335
5.50.2.3 AK_rel_eq_collect_cond_attributes() . . . 335
5.50.2.4 AK_rel_eq_get_attributes() . . . 336
5.50.2.5 AK_rel_eq_is_subset() . . . 336
5.50.2.6 AK_rel_eq_projection() . . . 337
5.50.2.7 AK_rel_eq_projection_attributes() . . . 338
5.50.2.8 AK_rel_eq_projection_test() . . . 338
5.50.2.9 AK_rel_eq_remove_duplicates() . . . 339
5.51 opti/rel_eq_projection.h File Reference . . . 339
5.51.1 Detailed Description . . . 340
5.51.2 Function Documentation . . . 340
5.51.2.1 AK_print_rel_eq_projection() . . . 340
5.51.2.2 AK_rel_eq_can_commute() . . . 340
5.51.2.3 AK_rel_eq_collect_cond_attributes() . . . 341
5.51.2.4 AK_rel_eq_get_attributes() . . . 341
5.51.2.5 AK_rel_eq_is_subset() . . . 342
5.51.2.6 AK_rel_eq_projection() . . . 343
5.51.2.7 AK_rel_eq_projection_attributes() . . . 344
5.51.2.8 AK_rel_eq_projection_test() . . . 344
5.51.2.9 AK_rel_eq_remove_duplicates() . . . 345
5.52 opti/rel_eq_selection.c File Reference . . . 345
5.52.1 Detailed Description . . . 346
5.52.2 Function Documentation . . . 346
5.52.2.1 AK_print_rel_eq_selection() . . . 346
5.52.2.2 AK_rel_eq_cond_attributes() . . . 346
5.52.2.3 AK_rel_eq_get_atrributes_char() . . . 347
5.52.2.4 AK_rel_eq_is_attr_subset() . . . 347
5.52.2.5 AK_rel_eq_selection() . . . 348
5.52.2.6 AK_rel_eq_selection_test() . . . 348
5.52.2.7 AK_rel_eq_share_attributes() . . . 349
5.52.2.8 AK_rel_eq_split_condition() . . . 349
5.53 opti/rel_eq_selection.h File Reference . . . 350
5.53.1 Detailed Description . . . 351
5.53.2 Function Documentation . . . 351
5.53.2.1 AK_print_rel_eq_selection() . . . 351

5.53.2.2 AK_rel_eq_cond_attributes() . . . 352
5.53.2.3 AK_rel_eq_get_atrributes_char() . . . 352
5.53.2.4 AK_rel_eq_is_attr_subset() . . . 354
5.53.2.5 AK_rel_eq_selection() . . . 355
5.53.2.6 AK_rel_eq_selection_test() . . . 355
5.53.2.7 AK_rel_eq_share_attributes() . . . 355
5.53.2.8 AK_rel_eq_split_condition() . . . 356
5.54 rec/archive_log.h File Reference . . . 357
5.54.1 Detailed Description . . . 358
5.54.2 Function Documentation . . . 358
5.54.2.1 AK_archive_log() . . . 358
5.54.2.2 AK_get_timestamp() . . . 359
5.55 rec/recovery.c File Reference . . . 359
5.55.1 Detailed Description . . . 360
5.55.2 Function Documentation . . . 360
5.55.2.1 AK_load_chosen_log() . . . 360
5.55.2.2 AK_load_latest_log() . . . 360
5.55.2.3 AK_recover_archive_log() . . . 361
5.55.2.4 AK_recover_operation() . . . 361
5.55.2.5 AK_recovery_insert_row() . . . 362
5.55.2.6 AK_recovery_test() . . . 362
5.55.2.7 AK_recovery_tokenize() . . . 363
5.55.2.8 recovery_insert_row() . . . 363
5.55.3 Variable Documentation . . . 363
5.55.3.1 grandfailure . . . 364
5.56 rec/redo_log.c File Reference . . . 364
5.56.1 Detailed Description . . . 364
5.56.2 Function Documentation . . . 364
5.56.2.1 AK_add_to_redolog() . . . 364
5.56.2.2 AK_add_to_redolog_select() . . . 365
5.56.2.3 AK_check_attributes() . . . 365
5.56.2.4 AK_check_redo_log_select() . . . 365
5.56.2.5 AK_printout_redolog() . . . 366
5.57 rel/aggregation.c File Reference . . . 366
5.57.1 Detailed Description . . . 367
5.57.2 Function Documentation . . . 367
5.57.2.1 AK_agg_input_add() . . . 367
5.57.2.2 AK_agg_input_add_to_beginning() . . . 367
5.57.2.3 AK_agg_input_fix() . . . 368
5.57.2.4 AK_agg_input_init() . . . 368
5.57.2.5 AK_aggregation() . . . 369
5.57.2.6 AK_aggregation_test() . . . 370

5.57.2.7 AK_header_size() . . . 370
5.57.2.8 AK_search_unsorted() . . . 370
5.58 rel/aggregation.h File Reference . . . 371
5.58.1 Detailed Description . . . 372
5.58.2 Function Documentation . . . 372
5.58.2.1 AK_agg_input_add() . . . 372
5.58.2.2 AK_agg_input_add_to_beginning() . . . 373
5.58.2.3 AK_agg_input_fix() . . . 373
5.58.2.4 AK_agg_input_init() . . . 374
5.58.2.5 AK_aggregation() . . . 374
5.58.2.6 AK_aggregation_test() . . . 375
5.58.2.7 AK_header_size() . . . 375
5.59 rel/difference.c File Reference . . . 376
5.59.1 Detailed Description . . . 376
5.59.2 Function Documentation . . . 376
5.59.2.1 AK_difference() . . . 376
5.59.2.2 AK_op_difference_test() . . . 377
5.60 rel/difference.h File Reference . . . 377
5.60.1 Detailed Description . . . 377
5.60.2 Function Documentation . . . 377
5.60.2.1 AK_difference() . . . 378
5.60.2.2 AK_op_difference_test() . . . 378
5.61 rel/expression_check.c File Reference . . . 378
5.61.1 Detailed Description . . . 379
5.61.2 Function Documentation . . . 379
5.61.2.1 AK_check_arithmetic_statement() . . . 379
5.61.2.2 AK_check_if_row_satisfies_expression() . . . 380
5.61.2.3 AK_check_regex_expression() . . . 380
5.61.2.4 AK_check_regex_operator_expression() . . . 381
5.61.2.5 AK_replace_wild_card() . . . 381
5.62 rel/expression_check.h File Reference . . . 381
5.62.1 Detailed Description . . . 382
5.62.2 Function Documentation . . . 382
5.62.2.1 AK_check_arithmetic_statement() . . . 382
5.62.2.2 AK_check_if_row_satisfies_expression() . . . 383
5.62.2.3 AK_check_regex_expression() . . . 384
5.62.2.4 AK_check_regex_operator_expression() . . . 384
5.63 rel/intersect.c File Reference . . . 385
5.63.1 Detailed Description . . . 385
5.63.2 Function Documentation . . . 385
5.63.2.1 AK_intersect() . . . 385
5.63.2.2 AK_op_intersect_test() . . . 386

5.64 rel/intersect.h File Reference . . . 386
5.64.1 Detailed Description . . . 386
5.64.2 Function Documentation . . . 387
5.64.2.1 AK_intersect() . . . 387
5.64.2.2 AK_op_intersect_test() . . . 387
5.65 rel/nat_join.c File Reference . . . 387
5.65.1 Detailed Description . . . 388
5.65.2 Function Documentation . . . 388
5.65.2.1 AK_copy_blocks_join() . . . 388
5.65.2.2 AK_create_join_block_header() . . . 389
5.65.2.3 AK_join() . . . 389
5.65.2.4 AK_merge_block_join() . . . 390
5.65.2.5 AK_op_join_test() . . . 390
5.66 rel/nat_join.h File Reference . . . 391
5.66.1 Detailed Description . . . 391
5.66.2 Function Documentation . . . 391
5.66.2.1 AK_copy_blocks_join() . . . 391
5.66.2.2 AK_create_join_block_header() . . . 392
5.66.2.3 AK_join() . . . 392
5.66.2.4 AK_merge_block_join() . . . 393
5.66.2.5 AK_op_join_test() . . . 393
5.67 rel/product.c File Reference . . . 394
5.67.1 Detailed Description . . . 394
5.67.2 Function Documentation . . . 394
5.67.2.1 AK_op_product_test() . . . 394
5.67.2.2 AK_product() . . . 394
5.67.2.3 AK_product_procedure() . . . 395
5.68 rel/product.h File Reference . . . 395
5.68.1 Detailed Description . . . 396
5.68.2 Function Documentation . . . 396
5.68.2.1 AK_op_product_test() . . . 396
5.68.2.2 AK_product() . . . 396
5.68.2.3 AK_product_procedure() . . . 397
5.69 rel/projection.c File Reference . . . 397
5.69.1 Detailed Description . . . 398
5.69.2 Function Documentation . . . 398
5.69.2.1 AK_copy_block_projection() . . . 398
5.69.2.2 AK_create_block_header() . . . 399
5.69.2.3 AK_create_header_name() . . . 399
5.69.2.4 AK_determine_header_type() . . . 400
5.69.2.5 AK_get_operator() . . . 400
5.69.2.6 AK_op_projection_test() . . . 401

5.69.2.7 AK_perform_operation() . . . 401
5.69.2.8 AK_projection() . . . 401
5.69.2.9 AK_remove_substring() . . . 402
5.70 rel/projection.h File Reference . . . 402
5.70.1 Detailed Description . . . 403
5.70.2 Function Documentation . . . 403
5.70.2.1 AK_copy_block_projection() . . . 404
5.70.2.2 AK_create_block_header() . . . 404
5.70.2.3 AK_create_header_name() . . . 405
5.70.2.4 AK_determine_header_type() . . . 405
5.70.2.5 AK_get_operator() . . . 406
5.70.2.6 AK_op_projection_test() . . . 406
5.70.2.7 AK_perform_operation() . . . 407
5.70.2.8 AK_projection() . . . 407
5.70.2.9 AK_remove_substring() . . . 408
5.71 rel/selection.c File Reference . . . 408
5.71.1 Detailed Description . . . 409
5.71.2 Function Documentation . . . 409
5.71.2.1 AK_op_selection_test() . . . 409
5.71.2.2 AK_op_selection_test_pattern() . . . 409
5.71.2.3 AK_selection() . . . 409
5.71.2.4 AK_selection_op_rename() . . . 410
5.72 rel/selection.h File Reference . . . 410
5.72.1 Detailed Description . . . 411
5.72.2 Function Documentation . . . 411
5.72.2.1 AK_op_selection_test() . . . 411
5.72.2.2 AK_op_selection_test_pattern() . . . 411
5.72.2.3 AK_selection() . . . 411
5.73 rel/theta_join.c File Reference . . . 412
5.73.1 Detailed Description . . . 412
5.73.2 Function Documentation . . . 412
5.73.2.1 AK_check_constraints() . . . 412
5.73.2.2 AK_create_theta_join_header() . . . 413
5.73.2.3 AK_op_theta_join_test() . . . 413
5.73.2.4 AK_theta_join() . . . 414
5.74 rel/theta_join.h File Reference . . . 414
5.74.1 Detailed Description . . . 415
5.74.2 Function Documentation . . . 415
5.74.2.1 AK_check_constraints() . . . 415
5.74.2.2 AK_create_theta_join_header() . . . 416
5.74.2.3 AK_op_theta_join_test() . . . 416
5.74.2.4 AK_theta_join() . . . 417

5.75 rel/union.c File Reference . . . 417
5.75.1 Detailed Description . . . 418
5.75.2 Function Documentation . . . 418
5.75.2.1 AK_op_union_test() . . . 418
5.75.2.2 AK_union() . . . 418
5.76 rel/union.h File Reference . . . 419
5.76.1 Detailed Description . . . 419
5.76.2 Function Documentation . . . 419
5.76.2.1 AK_op_union_test() . . . 419
5.76.2.2 AK_union() . . . 420
5.77 sql/command.c File Reference . . . 420
5.77.1 Detailed Description . . . 421
5.77.2 Function Documentation . . . 421
5.77.2.1 AK_command() . . . 421
5.77.2.2 AK_test_command() . . . 421
5.78 sql/command.h File Reference . . . 421
5.78.1 Detailed Description . . . 422
5.78.2 Function Documentation . . . 422
5.78.2.1 AK_command() . . . 422
5.78.2.2 AK_test_command() . . . 423
5.79 sql/cs/between.c File Reference . . . 423
5.79.1 Detailed Description . . . 423
5.79.2 Function Documentation . . . 423
5.79.2.1 AK_constraint_between_test() . . . 424
5.79.2.2 AK_delete_constraint_between() . . . 424
5.79.2.3 AK_find_table_address() . . . 424
5.79.2.4 AK_print_constraints() . . . 425
5.79.2.5 AK_read_constraint_between() . . . 425
5.79.2.6 AK_set_constraint_between() . . . 426
5.80 sql/cs/between.h File Reference . . . 426
5.80.1 Detailed Description . . . 427
5.80.2 Function Documentation . . . 427
5.80.2.1 AK_constraint_between_test() . . . 427
5.80.2.2 AK_delete_constraint_between() . . . 427
5.80.2.3 AK_find_table_address() . . . 428
5.80.2.4 AK_read_constraint_between() . . . 428
5.80.2.5 AK_set_constraint_between() . . . 429
5.81 sql/cs/check_constraint.c File Reference . . . 430
5.81.1 Detailed Description . . . 430
5.81.2 Function Documentation . . . 430
5.81.2.1 AK_check_constraint() . . . 430
5.81.2.2 AK_check_constraint_test() . . . 431

5.81.2.3 AK_set_check_constraint() . . . 431
5.81.2.4 condition_passed() . . . 432
5.82 sql/cs/check_constraint.h File Reference . . . 432
5.82.1 Detailed Description . . . 433
5.82.2 Function Documentation . . . 433
5.82.2.1 AK_check_constraint() . . . 433
5.82.2.2 AK_check_constraint_test() . . . 433
5.82.2.3 AK_set_check_constraint() . . . 434
5.82.2.4 condition_passed() . . . 434
5.83 sql/cs/constraint_names.c File Reference . . . 435
5.83.1 Detailed Description . . . 435
5.83.2 Function Documentation . . . 435
5.83.2.1 AK_check_constraint_name() . . . 435
5.83.2.2 AK_constraint_names_test() . . . 436
5.84 sql/cs/constraint_names.h File Reference . . . 436
5.84.1 Detailed Description . . . 436
5.84.2 Function Documentation . . . 436
5.84.2.1 AK_check_constraint_name() . . . 436
5.84.2.2 AK_constraint_names_test() . . . 437
5.85 sql/cs/nnull.c File Reference . . . 437
5.85.1 Detailed Description . . . 438
5.85.2 Function Documentation . . . 438
5.85.2.1 AK_check_constraint_not_null() . . . 438
5.85.2.2 AK_delete_constraint_not_null() . . . 439
5.85.2.3 AK_nnull_constraint_test() . . . 439
5.85.2.4 AK_read_constraint_not_null() . . . 439
5.85.2.5 AK_set_constraint_not_null() . . . 440
5.86 sql/cs/nnull.h File Reference . . . 440
5.86.1 Detailed Description . . . 441
5.86.2 Function Documentation . . . 441
5.86.2.1 AK_check_constraint_not_null() . . . 441
5.86.2.2 AK_delete_constraint_not_null() . . . 441
5.86.2.3 AK_nnull_constraint_test() . . . 442
5.86.2.4 AK_read_constraint_not_null() . . . 442
5.86.2.5 AK_set_constraint_not_null() . . . 443
5.87 sql/cs/reference.c File Reference . . . 443
5.87.1 Detailed Description . . . 444
5.87.2 Function Documentation . . . 444
5.87.2.1 AK_add_reference() . . . 444
5.87.2.2 AK_get_reference() . . . 445
5.87.2.3 AK_reference_check_attribute() . . . 445
5.87.2.4 AK_reference_check_entry() . . . 446

5.87.2.5 AK_reference_check_if_update_needed() . . . 446
5.87.2.6 AK_reference_check_restricion() . . . 447
5.87.2.7 AK_reference_test() . . . 447
5.87.2.8 AK_reference_update() . . . 447
5.88 sql/cs/reference.h File Reference . . . 448
5.88.1 Detailed Description . . . 449
5.88.2 Macro Definition Documentation . . . 450
5.88.2.1 REF_TYPE_NO_ACTION . . . 450
5.88.3 Function Documentation . . . 450
5.88.3.1 AK_add_reference() . . . 450
5.88.3.2 AK_delete_row() . . . 451
5.88.3.3 AK_get_reference() . . . 451
5.88.3.4 AK_initialize_new_segment() . . . 451
5.88.3.5 AK_Insert_New_Element() . . . 452
5.88.3.6 AK_Insert_New_Element_For_Update() . . . 452
5.88.3.7 AK_insert_row() . . . 453
5.88.3.8 AK_reference_check_attribute() . . . 454
5.88.3.9 AK_reference_check_entry() . . . 454
5.88.3.10 AK_reference_check_if_update_needed() . . . 455
5.88.3.11 AK_reference_check_restricion() . . . 455
5.88.3.12 AK_reference_test() . . . 456
5.88.3.13 AK_reference_update() . . . 456
5.88.3.14 AK_selection() . . . 456
5.88.3.15 AK_Update_Existing_Element() . . . 457
5.88.3.16 AK_update_row() . . . 458
5.89 sql/cs/unique.c File Reference . . . 458
5.89.1 Detailed Description . . . 458
5.89.2 Function Documentation . . . 458
5.89.2.1 AK_delete_constraint_unique() . . . 459
5.89.2.2 AK_read_constraint_unique() . . . 459
5.89.2.3 AK_set_constraint_unique() . . . 460
5.89.2.4 AK_unique_test() . . . 460
5.90 sql/cs/unique.h File Reference . . . 461
5.90.1 Detailed Description . . . 461
5.90.2 Function Documentation . . . 461
5.90.2.1 AK_delete_constraint_unique() . . . 461
5.90.2.2 AK_read_constraint_unique() . . . 462
5.90.2.3 AK_set_constraint_unique() . . . 463
5.90.2.4 AK_unique_test() . . . 463
5.91 sql/drop.c File Reference . . . 463
5.91.1 Detailed Description . . . 464
5.91.2 Function Documentation . . . 464

5.91.2.1 AK_drop() . . . 464
5.91.2.2 AK_drop_help_function() . . . 465
5.91.2.3 AK_drop_test() . . . 465
5.91.2.4 AK_if_exist() . . . 465
5.91.3 Variable Documentation . . . 466
5.91.3.1 system_catalog . . . 466
5.92 sql/drop.h File Reference . . . 466
5.92.1 Detailed Description . . . 467
5.92.2 Function Documentation . . . 467
5.92.2.1 AK_drop() . . . 467
5.92.2.2 AK_drop_test() . . . 468
5.92.2.3 AK_if_exist() . . . 468
5.93 sql/function.c File Reference . . . 468
5.93.1 Detailed Description . . . 469
5.93.2 Function Documentation . . . 469
5.93.2.1 AK_check_function_arguments() . . . 469
5.93.2.2 AK_check_function_arguments_type() . . . 470
5.93.2.3 AK_function_add() . . . 470
5.93.2.4 AK_function_arguments_add() . . . 471
5.93.2.5 AK_function_arguments_remove_by_obj_id() . . . 471
5.93.2.6 AK_function_change_return_type() . . . 472
5.93.2.7 AK_function_remove_by_name() . . . 472
5.93.2.8 AK_function_remove_by_obj_id() . . . 473
5.93.2.9 AK_function_rename() . . . 473
5.93.2.10 AK_function_test() . . . 474
5.93.2.11 AK_get_function_obj_id() . . . 474
5.94 sql/function.h File Reference . . . 474
5.94.1 Detailed Description . . . 475
5.94.2 Function Documentation . . . 475
5.94.2.1 AK_check_function_arguments() . . . 475
5.94.2.2 AK_check_function_arguments_type() . . . 476
5.94.2.3 AK_function_add() . . . 476
5.94.2.4 AK_function_arguments_add() . . . 477
5.94.2.5 AK_function_arguments_remove_by_obj_id() . . . 477
5.94.2.6 AK_function_change_return_type() . . . 478
5.94.2.7 AK_function_remove_by_name() . . . 478
5.94.2.8 AK_function_remove_by_obj_id() . . . 479
5.94.2.9 AK_function_rename() . . . 479
5.94.2.10 AK_function_test() . . . 480
5.94.2.11 AK_get_function_obj_id() . . . 480
5.95 sql/insert.h File Reference . . . 480
5.95.1 Detailed Description . . . 481

5.95.2 Function Documentation . . . 481
5.95.2.1 AK_get_insert_header() . . . 481
5.95.2.2 AK_insert() . . . 482
5.96 sql/privileges.c File Reference . . . 482
5.96.1 Detailed Description . . . 483
5.96.2 Function Documentation . . . 483
5.96.2.1 AK_add_user_to_group() . . . 483
5.96.2.2 AK_check_group_privilege() . . . 484
5.96.2.3 AK_check_privilege() . . . 484
5.96.2.4 AK_check_user_privilege() . . . 485
5.96.2.5 AK_grant_privilege_group() . . . 485
5.96.2.6 AK_grant_privilege_user() . . . 486
5.96.2.7 AK_group_add() . . . 486
5.96.2.8 AK_group_get_id() . . . 487
5.96.2.9 AK_group_remove_by_name() . . . 487
5.96.2.10 AK_group_rename() . . . 488
5.96.2.11 AK_privileges_test() . . . 488
5.96.2.12 AK_remove_all_users_from_group() . . . 488
5.96.2.13 AK_remove_user_from_all_groups() . . . 489
5.96.2.14 AK_revoke_all_privileges_group() . . . 489
5.96.2.15 AK_revoke_all_privileges_user() . . . 490
5.96.2.16 AK_revoke_privilege_group() . . . 490
5.96.2.17 AK_revoke_privilege_user() . . . 491
5.96.2.18 AK_user_add() . . . 491
5.96.2.19 AK_user_check_pass() . . . 492
5.96.2.20 AK_user_get_id() . . . 492
5.96.2.21 AK_user_remove_by_name() . . . 493
5.96.2.22 AK_user_rename() . . . 493
5.97 sql/privileges.h File Reference . . . 493
5.97.1 Detailed Description . . . 495
5.97.2 Function Documentation . . . 495
5.97.2.1 AK_add_user_to_group() . . . 495
5.97.2.2 AK_check_group_privilege() . . . 495
5.97.2.3 AK_check_privilege() . . . 496
5.97.2.4 AK_check_user_privilege() . . . 496
5.97.2.5 AK_grant_privilege_group() . . . 497
5.97.2.6 AK_grant_privilege_user() . . . 497
5.97.2.7 AK_group_add() . . . 498
5.97.2.8 AK_group_get_id() . . . 498
5.97.2.9 AK_group_remove_by_name() . . . 499
5.97.2.10 AK_group_rename() . . . 499
5.97.2.11 AK_privileges_test() . . . 500

5.97.2.12 AK_remove_all_users_from_group() . . . 500
5.97.2.13 AK_remove_user_from_all_groups() . . . 500
5.97.2.14 AK_revoke_all_privileges_group() . . . 501
5.97.2.15 AK_revoke_all_privileges_user() . . . 501
5.97.2.16 AK_revoke_privilege_group() . . . 502
5.97.2.17 AK_revoke_privilege_user() . . . 503
5.97.2.18 AK_user_add() . . . 503
5.97.2.19 AK_user_check_pass() . . . 504
5.97.2.20 AK_user_get_id() . . . 504
5.97.2.21 AK_user_rename() . . . 505
5.98 sql/select.c File Reference . . . 505
5.98.1 Detailed Description . . . 506
5.98.2 Function Documentation . . . 506
5.98.2.1 AK_select() . . . 506
5.98.2.2 AK_select_test() . . . 507
5.99 sql/select.h File Reference . . . 507
5.99.1 Detailed Description . . . 507
5.99.2 Function Documentation . . . 507
5.99.2.1 AK_select() . . . 507
5.99.2.2 AK_select_test() . . . 508
5.100 sql/trigger.c File Reference . . . 508
5.100.1 Detailed Description . . . 509
5.100.2 Function Documentation . . . 509
5.100.2.1 AK_trigger_add() . . . 509
5.100.2.2 AK_trigger_edit() . . . 509
5.100.2.3 AK_trigger_get_conditions() . . . 510
5.100.2.4 AK_trigger_get_id() . . . 510
5.100.2.5 AK_trigger_remove_by_name() . . . 511
5.100.2.6 AK_trigger_remove_by_obj_id() . . . 511
5.100.2.7 AK_trigger_rename() . . . 512
5.100.2.8 AK_trigger_save_conditions() . . . 512
5.100.2.9 AK_trigger_test() . . . 513
5.101 sql/trigger.h File Reference . . . 513
5.101.1 Detailed Description . . . 514
5.101.2 Function Documentation . . . 514
5.101.2.1 AK_trigger_add() . . . 514
5.101.2.2 AK_trigger_edit() . . . 514
5.101.2.3 AK_trigger_get_conditions() . . . 515
5.101.2.4 AK_trigger_get_id() . . . 516
5.101.2.5 AK_trigger_remove_by_name() . . . 516
5.101.2.6 AK_trigger_remove_by_obj_id() . . . 517
5.101.2.7 AK_trigger_rename() . . . 517

5.101.2.8 AK_trigger_save_conditions() . . . 518
5.101.2.9 AK_trigger_test() . . . 518
5.102 sql/view.c File Reference . . . 518
5.102.1 Detailed Description . . . 519
5.102.2 Function Documentation . . . 519
5.102.2.1 AK_check_view_name() . . . 519
5.102.2.2 AK_get_rel_exp() . . . 520
5.102.2.3 AK_get_view_obj_id() . . . 520
5.102.2.4 AK_get_view_query() . . . 520
5.102.2.5 AK_test_get_view_data() . . . 521
5.102.2.6 AK_view_add() . . . 521
5.102.2.7 AK_view_change_query() . . . 522
5.102.2.8 AK_view_remove_by_name() . . . 522
5.102.2.9 AK_view_remove_by_obj_id() . . . 523
5.102.2.10 AK_view_rename() . . . 523
5.102.2.11 AK_view_test() . . . 524
5.103 tools/comments.py File Reference . . . 524
5.103.1 Detailed Description . . . 524
5.104 tools/getFiles.sh File Reference . . . 524
5.104.1 Detailed Description . . . 524
5.105 tools/parseC.sh File Reference . . . 524
5.105.1 Detailed Description . . . 524
5.106 tools/parsePy.sh File Reference . . . 525
5.106.1 Detailed Description . . . 525
5.107 tools/updateVersion.sh File Reference . . . 525
5.107.1 Detailed Description . . . 525
5.108 trans/transaction.c File Reference . . . 525
5.108.1 Detailed Description . . . 527
5.108.2 Function Documentation . . . 527
5.108.2.1 AK_acquire_lock() . . . 527
5.108.2.2 AK_add_hash_entry_list() . . . 528
5.108.2.3 AK_add_lock() . . . 528
5.108.2.4 AK_all_transactions_finished() . . . 529
5.108.2.5 AK_create_lock() . . . 529
5.108.2.6 AK_create_new_transaction_thread() . . . 529
5.108.2.7 AK_delete_hash_entry_list() . . . 530
5.108.2.8 AK_delete_lock_entry_list() . . . 530
5.108.2.9 AK_execute_commands() . . . 531
5.108.2.10 AK_execute_transaction() . . . 531
5.108.2.11 AK_get_memory_blocks() . . . 532
5.108.2.12 AK_handle_observable_transaction_action() . . . 532
5.108.2.13 AK_init_observable_transaction() . . . 533

5.108.2.14 AK_init_observer_lock() . . . 533
5.108.2.15 AK_isLock_waiting() . . . 533
5.108.2.16 AK_lock_released() . . . 534
5.108.2.17 AK_memory_block_hash() . . . 534
5.108.2.18 AK_on_all_transactions_end() . . . 535
5.108.2.19 AK_on_lock_release() . . . 535
5.108.2.20 AK_on_observable_notify() . . . 535
5.108.2.21 AK_on_transaction_end() . . . 536
5.108.2.22 AK_release_locks() . . . 536
5.108.2.23 AK_remove_transaction_thread() . . . 536
5.108.2.24 AK_search_empty_link_for_hook() . . . 537
5.108.2.25 AK_search_existing_link_for_hook() . . . 537
5.108.2.26 AK_search_lock_entry_list_by_key() . . . 538
5.108.2.27 AK_transaction_finished() . . . 538
5.108.2.28 AK_transaction_manager() . . . 538
5.108.2.29 AK_transaction_register_observer() . . . 539
5.108.2.30 AK_transaction_unregister_observer() . . . 539
5.108.2.31 handle_transaction_notify() . . . 540
5.109 trans/transaction.h File Reference . . . 540
5.109.1 Detailed Description . . . 543
5.109.2 Enumeration Type Documentation . . . 543
5.109.2.1 NoticeType . . . 543
5.109.3 Function Documentation . . . 543
5.109.3.1 AK_acquire_lock() . . . 543
5.109.3.2 AK_add_hash_entry_list() . . . 544
5.109.3.3 AK_add_lock() . . . 545
5.109.3.4 AK_all_transactions_finished() . . . 545
5.109.3.5 AK_create_lock() . . . 545
5.109.3.6 AK_create_new_transaction_thread() . . . 546
5.109.3.7 AK_delete_hash_entry_list() . . . 546
5.109.3.8 AK_delete_lock_entry_list() . . . 547
5.109.3.9 AK_execute_commands() . . . 547
5.109.3.10 AK_execute_transaction() . . . 548
5.109.3.11 AK_get_memory_blocks() . . . 548
5.109.3.12 AK_handle_observable_transaction_action() . . . 549
5.109.3.13 AK_init_observable_transaction() . . . 549
5.109.3.14 AK_init_observer_lock() . . . 550
5.109.3.15 AK_isLock_waiting() . . . 550
5.109.3.16 AK_lock_released() . . . 550
5.109.3.17 AK_memory_block_hash() . . . 551
5.109.3.18 AK_on_all_transactions_end() . . . 552
5.109.3.19 AK_on_lock_release() . . . 552

5.109.3.20 AK_on_observable_notify() . . . 552
5.109.3.21 AK_on_transaction_end() . . . 552
5.109.3.22 AK_release_locks() . . . 553
5.109.3.23 AK_remove_transaction_thread() . . . 553
5.109.3.24 AK_search_empty_link_for_hook() . . . 554
5.109.3.25 AK_search_existing_link_for_hook() . . . 554
5.109.3.26 AK_search_lock_entry_list_by_key() . . . 555
5.109.3.27 AK_transaction_finished() . . . 555
5.109.3.28 AK_transaction_manager() . . . 555
5.109.3.29 AK_transaction_register_observer() . . . 556
5.109.3.30 AK_transaction_unregister_observer() . . . 556
5.109.3.31 handle_transaction_notify() . . . 557

Index **559**

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.

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

Member AK_acquire_lock (int, int, pthread_t)

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

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

Member AK_archive_log (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

Check multithreading, check if it's working correctly

Member AK_execute_commands (command *, int)

Check multithreading, check if it's working correctly

Check multithreading, check if it's working correctly

Member AK_get_timestamp ()

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

Member AK_memory_block_hash (int)

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

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

Member AK_memory_block_hash (int)

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

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

Member AK_sort_segment (char *srcTable, char *destTable, struct list_node *attributes)

Make it to suport multiple sort atributes and ASC|DESC ordering

Make it to suport multiple sort atributes and ASC|DESC ordering

Chapter 2

Class Index

2.1 Class List

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

dictionary
Dictionary object . . . 11
_file_metadata . . . 12
_notifyDetails . . . 12
AK_agg_input
Structure that contains attributes from table header, tasks for this table and counter value . . . 13
AK_agg_value
Structure that contains atribute name, date and aggregation task associated . . . 13
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 . . . 14
AK_block_activity
Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked_for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and uninteruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it . . . 14
AK_blocktable . . . 15
AK_command_recovery_struct
Recovery structure used to recover commands from binary file . . . 15
AK_command_struct . . . 16
AK_create_table_struct . . . 16
AK_db_cache
Structure that defines global cache memory . . . 17
AK_debmod_state
Global structure that holds all relevant information for the debug mode and related functionality 17
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 . . . 18

AK_mem_block
Structure that defines a block of data in memory . . . 19
AK_operand . . . 19
AK_query_mem
Structure that defines global query memory . . . 20
AK_query_mem_dict
Structure that defines global query memory for data dictionaries . . . 20
AK_query_mem_lib
Structure that defines global query memory for libraries . . . 21
AK_query_mem_result
Structure that defines global query memory for results . . . 21
AK_redo_log
Structure that defines global redo log . . . 22
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 . . . 23
AK_results
Structure used for in-memory result caching . . . 23
AK_synchronization_info
Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex) . . . 24
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 . . . 24
blocktable
Structure that defines bit status of blocks, last initialized and last allocated index . . . 25
btree_node . . . 25
bucket_elem
Structure for defining a single bucket element . . . 26
cost_eval_t
Stucture for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name) . . . 26
DEBUG_LEVEL
Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0 . . . 27
DEBUG_TYPE
Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be aditional added to the enum. Each debug type can be easly excluded from output by setting corresponding enum element to 0 . . . 27
drop_arguments . . . 28
hash_bucket
Structure for hash bucket for table hashing . . . 28
hash_info
Structure for defining a hash info element . . . 29
intersect_attr
Structure defines intersect attribute . . . 29
list_node
Structure defines a list node . . . 30
list_structure_ad . . . 31
list_structure_add
Structure that defines linked list node for index . . . 31
main_bucket
Structure for defining main bucket for table hashing . . . 31
memoryAddresses
Structure that represents a linked list of locked addresses . . . 32
Observable
Structure that defines the functions for observable object . . . 32

observable_transaction
Structure which defines transaction observable type . . . 33
observable_transaction_struct . . . 33
Observer
Structure that defines the functions for observer object . . . 34
observer_lock
Structure which defines transaction lock observer type . . . 34
root_info . . . 35
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 . . . 35
search_result
Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_↩ unsorted . . . 36
Stack
Structure defines a Stack element. Every Stack has its Vertex pointer and pointer to next Stack in the linked list . . . 37
struct_add
Structure defining node address . . . 37
Succesor
Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor in the linked list . . . 38
table_addresses
Structure that defines start and end address of extent . . . 38
TestResult
Used so tests can report the amount of successful tests . . . 39
threadContainer
Structure that represents a linked list of threads.
39
transaction_list_elem
Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table . . . 40
transaction_list_head
Structure that represents LockTable entry about doubly linked list of collision in Hash table . . . 41
transaction_locks_list_elem
Structure that represents LockTable entry about transaction resource lock . . . 41
transactionData
Structure used to transport transaction data to the thread . . . 42
TypeObservable . . . 42
TypeObserver . . . 42
Vertex
Structure defines a Vertex node element. Every Vertex has its VertexId, index, lowLink and pointer to next edge and vertex . . . 43

Chapter 3

File Index

3.1 File List

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

auxi/auxiliary.c . . . 45
auxi/auxiliary.h . . . 45
auxi/**configuration.h** . . . **??**
auxi/constants.h . . . 64
auxi/debug.c . . . 69
auxi/debug.h . . . 70
auxi/dictionary.c
    Implements a dictionary for string variables . . . 72
auxi/dictionary.h
    Implements a dictionary for string variables . . . 76
auxi/iniparser.c
    Parser for ini files . . . 80
auxi/iniparser.h
    Parser for ini files . . . 90
auxi/mempro.c . . . 99
auxi/mempro.h . . . 114
auxi/observable.c . . . 128
auxi/observable.h . . . 131
auxi/test.c . . . 132
auxi/**test.h** . . . **??**
dm/dbman.c . . . 142
dm/dbman.h . . . 159
file/blobs.c . . . 179
file/blobs.h . . . 184
file/fileio.c . . . 188
file/fileio.h . . . 194
file/files.c . . . 200
file/files.h . . . 202
file/filesearch.c . . . 204
file/filesearch.h . . . 206
file/filesort.h . . . 209
file/id.c . . . 212
file/id.h . . . 214
file/sequence.c . . . 262
file/sequence.h . . . 267

file/table.c . . . 272
file/table.h . . . 285
file/**tableOld.h** . . . **??**
file/test.c . . . 134
file/test.h . . . 138
file/idx/bitmap.c . . . 215
file/idx/bitmap.h . . . 222
file/idx/btree.c . . . 230
file/idx/btree.h . . . 231
file/idx/hash.c . . . 233
file/idx/hash.h . . . 239
file/idx/index.c . . . 246
file/idx/index.h . . . 254
mm/memoman.c . . . 298
mm/memoman.h . . . 308
opti/query_optimization.c . . . 319
opti/query_optimization.h . . . 321
opti/rel_eq_assoc.c . . . 324
opti/rel_eq_assoc.h . . . 326
opti/rel_eq_comut.c . . . 329
opti/rel_eq_comut.h . . . 331
opti/rel_eq_projection.c . . . 333
opti/rel_eq_projection.h . . . 339
opti/rel_eq_selection.c . . . 345
opti/rel_eq_selection.h . . . 350
rec/archive_log.h . . . 357
rec/recovery.c . . . 359
rec/**recovery.h** . . . **??**
rec/redo_log.c . . . 364
rec/**redo_log.h** . . . **??**
rel/aggregation.c . . . 366
rel/aggregation.h . . . 371
rel/difference.c . . . 376
rel/difference.h . . . 377
rel/expression_check.c . . . 378
rel/expression_check.h . . . 381
rel/intersect.c . . . 385
rel/intersect.h . . . 386
rel/nat_join.c . . . 387
rel/nat_join.h . . . 391
rel/product.c . . . 394
rel/product.h . . . 395
rel/projection.c . . . 397
rel/projection.h . . . 402
rel/selection.c . . . 408
rel/selection.h . . . 410
rel/theta_join.c . . . 412
rel/theta_join.h . . . 414
rel/union.c . . . 417
rel/union.h . . . 419
sql/command.c . . . 420
sql/command.h . . . 421
sql/drop.c . . . 463
sql/drop.h . . . 466
sql/function.c . . . 468
sql/function.h . . . 474
sql/insert.h . . . 480
sql/privileges.c . . . 482

sql/privileges.h . . . 493
sql/select.c . . . 505
sql/select.h . . . 507
sql/trigger.c . . . 508
sql/trigger.h . . . 513
sql/view.c . . . 518
sql/**view.h** . . . **??**
sql/cs/between.c . . . 423
sql/cs/between.h . . . 426
sql/cs/check_constraint.c . . . 430
sql/cs/check_constraint.h . . . 432
sql/cs/constraint_names.c . . . 435
sql/cs/constraint_names.h . . . 436
sql/cs/nnull.c . . . 437
sql/cs/nnull.h . . . 440
sql/cs/reference.c . . . 443
sql/cs/reference.h . . . 448
sql/cs/unique.c . . . 458
sql/cs/unique.h . . . 461
tools/comments.py . . . 524
tools/getFiles.sh . . . 524
tools/parseC.sh . . . 524
tools/parsePy.sh . . . 525
tools/updateVersion.sh . . . 525
trans/transaction.c . . . 525
trans/transaction.h . . . 540

Chapter 4

Class Documentation

4.1 _dictionary_ Struct Reference

Dictionary object.

`#include <dictionary.h>`

Public Attributes

- int **n**
- int size
- char ** val
- char ** key
- unsigned * hash

4.1.1 Detailed Description

Dictionary object.

This object contains a list of string/string associations. Each association is identified by a unique string key. Looking up values in the dictionary is speeded up by the use of a (hopefully collision-AK_free) hash function.

4.1.2 Member Data Documentation

4.1.2.1 hash

`unsigned* _dictionary_::hash`

List of string keys

4.1.2.2 key

```
char** _dictionary_::key
```

List of string values

4.1.2.3 size

```
int _dictionary_::size
```

Number of entries in dictionary

4.1.2.4 val

```
char** _dictionary_::val
```

Storage size

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

- auxi/dictionary.h

4.2 _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

4.3 _notifyDetails Struct Reference

Public Attributes

- char * **message**
- NotifyType **type**

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

- auxi/observable.c

4.4 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**

4.4.1 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

4.5 AK_agg_value Struct Reference

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

`#include <aggregation.h>`

Public Attributes

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

4.5.1 Detailed Description

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

Author

Unknown

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

- rel/aggregation.h

4.6 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

4.6.1 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

4.7 AK_block_activity Struct Reference

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

`#include <dbman.h>`

Public Attributes

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

4.7.1 Detailed Description

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

Author

Domagoj Šitum

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

- dm/dbman.h

4.8 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

4.9 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]
- char **condition** [MAX_ATTRIBUTES][MAX_VARCHAR_LENGTH]
- int **finished**

4.9.1 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. Updated can be used to save select operation

Author

Tomislav Turek updated by Danko Bukovac

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

- mm/memoman.h

4.10 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

4.11 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 files:

- file/table.h
- file/tableOld.h

4.12 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

4.12.1 Detailed Description

Structure that defines global cache memory.

Author

Unknown

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

- mm/memoman.h

4.13 AK_debmod_state Struct Reference

Global structure that holds all relevant information for the debug mode and related functionality.

`#include <mempro.h>`

Public Attributes

- uint8_t **init**
- uint32_t **page_size**
- uint8_t **ready**
- void * **page** [AK_DEBMOD_PAGES_NUM]
- uint8_t **used** [AK_DEBMOD_PAGES_NUM]
- uint32_t **nomi** [AK_DEBMOD_PAGES_NUM]
- uint32_t **real** [AK_DEBMOD_PAGES_NUM]
- uint8_t **dirty** [AK_DEBMOD_PAGES_NUM]
- char **function** [AK_DEBMOD_MAX_FUNCTIONS][AK_DEBMOD_MAX_FUNC_NAME]
- int32_t **last_function_id**
- int32_t **alloc_owner** [AK_DEBMOD_PAGES_NUM]
- int32_t **free_owner** [AK_DEBMOD_PAGES_NUM]
- int8_t **func_used_by** [AK_DEBMOD_MAX_FUNCTIONS][AK_DEBMOD_MAX_FUNCTIONS]
- uint8_t **print**
- int32_t **fstack_size**
- int32_t **fstack_items** [AK_DEBMOD_STACKSIZE]

4.13.1 Detailed Description

Global structure that holds all relevant information for the debug mode and related functionality.

Author

Marin Rukavina, Mislav Bozicevic

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

- auxi/mempro.h

4.14 AK_header Struct Reference

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

`#include <dbman.h>`

Public Attributes

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

4.14.1 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

4.15 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

4.15.1 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

4.16 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

4.17 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

4.17.1 Detailed Description

Structure that defines global query memory.

Author

Unknown

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

- mm/memoman.h

4.18 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)

4.18.1 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

4.19 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)

4.19.1 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

4.20 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)

4.20.1 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

4.21 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**

4.21.1 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

4.22 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**

4.22.1 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

4.23 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]

4.23.1 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

4.24 AK_synchronization_info Struct Reference

Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

`#include <auxiliary.h>`

Public Attributes

- int **init**
- int **ready**

4.24.1 Detailed Description

Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

Author

Marko Sinko

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

- auxi/auxiliary.h

4.25 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(***))*

4.25.1 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

4.26 blocktable Struct Reference

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

`#include <dbman.h>`

4.26.1 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

4.27 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

4.28 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

4.28.1 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

4.29 cost_eval_t Struct Reference

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

`#include <rel_eq_assoc.h>`

Public Attributes

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

4.29.1 Detailed Description

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

Author

Dino Laktašić

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

- opti/rel_eq_assoc.h

4.30 DEBUG_LEVEL Struct Reference

Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0.

```
#include <debug.h>
```

4.30.1 Detailed Description

Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0.

Author

Dino Laktašić

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

- auxi/debug.h

4.31 DEBUG_TYPE Struct Reference

Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be aditional added to the enum. Each debug type can be easly excluded from output by setting corresponding enum element to 0.

```
#include <debug.h>
```

4.31.1 Detailed Description

Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be aditional added to the enum. Each debug type can be easly excluded from output by setting corresponding enum element to 0.

Author

Dino Laktašić

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

- auxi/debug.h

4.32 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

4.33 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

4.33.1 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

4.34 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

4.34.1 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

4.35 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

4.35.1 Detailed Description

Structure defines intersect attribute.

Author

Dino Laktašić

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

- rel/intersect.h

4.36 list_node Struct Reference

Structure defines a list node.

`#include <auxiliary.h>`

Collaboration diagram for list_node:

Public Attributes

- int type
  TODO - type, attribute name, table staviti na početak polja data data type.
- int **size**
- char data [MAX_VARCHAR_LENGTH]
  loaded data
- char **table** [MAX_ATT_NAME]
- char **attribute_name** [MAX_ATT_NAME]
- int **constraint**
- struct list_node * next
  pointer to next element

4.36.1 Detailed Description

Structure defines a list node.

Author

Ljiljana Pintarić

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

- auxi/auxiliary.h

4.37 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

4.38 list_structure_add Struct Reference

Structure that defines linked list node for index.

`#include <index.h>`

4.38.1 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

4.39 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

4.39.1 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

4.40 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**

4.40.1 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

4.41 Observable Struct Reference

Structure that defines the functions for observable object.

`#include <observable.h>`

Collaboration diagram for Observable:

Public Attributes

- AK_observer * **observers** [MAX_OBSERVABLE_OBSERVERS]
- int **observer_id_counter**
- void * **AK_observable_type**
- int **AK_ObservableType_Def**
- int(* **AK_destroy_observable**)(struct Observable *)
- int(* **AK_register_observer**)(struct Observable *, AK_observer *)
- int(* **AK_unregister_observer**)(struct Observable *, AK_observer *)
- int(* **AK_notify_observer**)(struct Observable *, AK_observer *)
- int(* **AK_notify_observers**)(struct Observable *)
- int(* **AK_run_custom_action**)(void *)
- AK_observer *(* **AK_get_observer_by_id**)(struct Observable *, int id)

4.41.1 Detailed Description

Structure that defines the functions for observable object.

Author

Ivan Pusic

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

- auxi/observable.h

4.42 observable_transaction Struct Reference

Structure which defines transaction observable type.

`#include <transaction.h>`

4.42.1 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

4.43 observable_transaction_struct Struct Reference

Collaboration diagram for observable_transaction_struct:

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

4.44 Observer Struct Reference

Structure that defines the functions for observer object.

`#include <observable.h>`

Public Attributes

- int **observer_id**
- void ∗ **AK_observer_type**
- void(∗ **AK_observer_type_event_handler**)(void ∗, void ∗, AK_ObservableType_Enum)
- int(∗ **AK_notify**)(struct Observer ∗, void ∗observable_type, AK_ObservableType_Enum)
- int(∗ **AK_destroy_observer**)(struct Observer ∗)

4.44.1 Detailed Description

Structure that defines the functions for observer object.

Author

Ivan Pusic

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

- auxi/observable.h

4.45 observer_lock Struct Reference

Structure which defines transaction lock observer type.

`#include <transaction.h>`

Collaboration diagram for observer_lock:

Public Attributes

- AK_observer * **observer**

4.45.1 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

4.46 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

4.47 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

4.47.1 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

4.48 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

4.48.1 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

4.49 Stack Struct Reference

Structure defines a Stack element. Every Stack has its Vertex pointer and pointer to next Stack in the linked list.

`#include <auxiliary.h>`

Collaboration diagram for Stack:

Public Attributes

- struct Vertex * **link**
- struct Stack * **nextElement**

4.49.1 Detailed Description

Structure defines a Stack element. Every Stack has its Vertex pointer and pointer to next Stack in the linked list.

Author

Frane Jakelić

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

- auxi/auxiliary.h

4.50 struct_add Struct Reference

Structure defining node address.

`#include <index.h>`

Public Attributes

- int addBlock
  block address
- int indexTd
  index table destination

4.50.1 Detailed Description

Structure defining node address.

Author

Unknown

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

- file/idx/index.h

4.51 Succesor Struct Reference

Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor in the linked list.

`#include <auxiliary.h>`

Collaboration diagram for Succesor:

Public Attributes

- struct Vertex * **link**
- struct Succesor * **nextSuccesor**

4.51.1 Detailed Description

Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor in the linked list.

Author

Frane Jakelić

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

- auxi/auxiliary.h

4.52 table_addresses Struct Reference

Structure that defines start and end address of extent.

`#include <dbman.h>`

Public Attributes

- int address_from [MAX_EXTENTS_IN_SEGMENT]

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

4.52.1 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

4.53 TestResult Struct Reference

Used so tests can report the amount of successful tests.

`#include <test.h>`

Public Attributes

- int **testSucceded**
- int **testFailed**
- char **implemented**

4.53.1 Detailed Description

Used so tests can report the amount of successful tests.

This structure is used so tests can report the amount of successful tests.

Author

Igor Rinkovec

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

- auxi/test.h

4.54 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**

4.54.1 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

4.55 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**

4.55.1 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

4.56 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**

4.56.1 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

4.57 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**

4.57.1 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

4.58 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**

4.58.1 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

4.59 TypeObservable Struct Reference

Collaboration diagram for TypeObservable:

Public Attributes

- NotifyDetails * **notifyDetails**
- char *(* **AK_get_message**)(struct TypeObservable *)
- int(* **AK_custom_register_observer**)(struct TypeObservable *, AK_observer *)
- int(* **AK_custom_unregister_observer**)(struct TypeObservable *, AK_observer *)
- void(* **AK_set_notify_info_details**)(struct TypeObservable *, NotifyType type, char *message)
- AK_observable * **observable**

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

- auxi/observable.c

4.60 TypeObserver Struct Reference

Collaboration diagram for TypeObserver:

Public Attributes

- AK_TypeObservable * **observable**
- AK_observer * **observer**

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

- auxi/observable.c

4.61 Vertex Struct Reference

Structure defines a Vertex node element. Every Vertex has its VertexId, index, lowLink and pointer to next edge and vertex.

```
#include <auxiliary.h>
```

Collaboration diagram for Vertex:

Public Attributes

- int **vertexId**
- int **index**
- int **lowLink**
- struct Succesor * **nextSuccesor**
- struct Vertex * **nextVertex**

4.61.1 Detailed Description

Structure defines a Vertex node element. Every Vertex has its VertexId, index, lowLink and pointer to next edge and vertex.

Author

Frane Jakelić

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

- auxi/auxiliary.h

Chapter 5

File Documentation

5.1 auxi/auxiliary.c File Reference

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

5.2 auxi/auxiliary.h File Reference

```
#include "constants.h"
#include "configuration.h"
#include "test.h"
#include "assert.h"
#include "time.h"
#include "string.h"
#include "ctype.h"
#include "debug.h"
#include "mempro.h"
```
Include dependency graph for auxiliary.h: This graph shows which files directly or indirectly include this file:

Classes

- struct list_node

  Structure defines a list node.
- struct Vertex

  Structure defines a Vertex node element. Every Vertex has its VertexId, index, lowLink and pointer to next edge and vertex.
- struct Succesor

  Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor in the linked list.
- struct Stack

  Structure defines a Stack element. Every Stack has its Vertex pointer and pointer to next Stack in the linked list.
- struct AK_synchronization_info

  Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

Macros

- #define **MAX_LOOP_ITERATIONS** 1000
- #define **TBL_BOX_OFFSET** 1

Typedefs

- typedef struct list_node **AK_list**
- typedef struct list_node * **AK_list_elem**
- typedef struct Vertex **AK_graph**
- typedef struct Succesor * **AK_succesor**
- typedef struct Vertex * **AK_vertex**
- typedef struct Stack * **AK_stack**
- typedef struct Stack **AK_stackHead**

Functions

- char * AK_convert_type (char *arg_type)

  Function that change type of argument from string to integer.
- int AK_strcmp (const void *a, const void *b)

  Function compares two Strings.
- int AK_chars_num_from_number (int number, int base)

  Function that gets the number of digits for any given number.
- size_t AK_type_size (int iDB_type, char *szVarchar)

  Function returns the size in bytes for the provided database type.
- void AK_Init_L3 (struct list_node **L)

  Function that initializes an empty list.
- struct list_node * AK_First_L2 (struct list_node *L)

  Function that fetches the first element of the list.
- struct list_node * AK_End_L2 (struct list_node *L)

  Function that fetches the last element of the list.
- struct list_node * AK_Next_L2 (struct list_node *current)

  Function that fetches the next element of the list.
- struct list_node * AK_Previous_L2 (struct list_node *current, struct list_node *L)

  Function that fetches the previous element of the list.
- unsigned int AK_IsEmpty_L2 (struct list_node *L)

  Function that tests if the list is empty.
- void AK_InsertBefore_L2 (int type, char *data, int size, struct list_node **current, struct list_node **L)

  Function that inserts a new element before the current element of the list.
- void AK_InsertAfter_L2 (int type, char *data, int size, struct list_node **current, struct list_node **L)

  Function that inserts a new element after the current element of the list.
- void AK_InsertAtBegin_L3 (int type, char *data, int size, struct list_node *L)

  Function that inserts a new element at the beginning of the list. It uses function called: AK_InsertBefore_L.
- void AK_InsertAtEnd_L3 (int type, char *data, int size, struct list_node *L)

  Function that inserts a new element at the end of the list. It uses a function called: AK_InsertAfter_L2.
- void AK_Delete_L3 (struct list_node **current, struct list_node **L)

  Function that deletes the current element of the list.
- void AK_DeleteAll_L3 (struct list_node **L)

  Function that empties the list.
- int AK_Size_L2 (struct list_node *L)

Function that fetches the number of the elements in the list.

- char * AK_Retrieve_L2 (struct list_node *current, struct list_node *L)

  Function that retrieves the data from the current element of the list.
- struct list_node * AK_GetNth_L2 (int pos, struct list_node *row)

  Function that fetches the nth element in a row.
- char * AK_get_array_perms (char *arr)

  Get all permutations without repetition (currently not used, but it can be helpful)
- AK_vertex AK_search_vertex (int id)

  Function that searches for a specific graph node by its ID.
- AK_vertex AK_search_empty_link ()

  Looks for empty link for a new graph node.
- AK_vertex AK_add_vertex (int id)

  Function that adds a new graph node.
- AK_succesor AK_add_succesor (int succesorId, int succesorOf)

  Creates an edge between two nodes.
- AK_stack AK_search_empty_stack_link (AK_stack stackRoot)

  Returns a empty link for the stack.
- AK_stack AK_push_to_stack (int id)

  Adds a entry to the stack.
- AK_stack AK_pop_from_stack ()

  Pops a entry to the stack.
- AK_stack AK_search_in_stack (int id)

  Finds an element in the stack.
- int **MIN** (int X, int Y)
- void AK_tarjan (int id)

  Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.
- TestResult **AK_tarjan_test** ()
- AK_synchronization_info * AK_init_critical_section ()

  Initializes an AK_synchronization_info structure and returns an owned pointer that must later be passed on to AK_destroy_critical_section.
- void AK_destroy_critical_section (AK_synchronization_info *info)

  Destroys a synchronization object when it is no longer necessary and frees the pointer.
- void AK_enter_critical_section (AK_synchronization_info *info)

  Enters a critical section.
- void AK_leave_critical_section (AK_synchronization_info *info)

  Leaves a critical section.

Variables

- int testMode

  You can turn testMode on or off with TEST_MODE_ON and TEST_MODE_OFF. To do this, simply enable or disable it in YOUR function (not in any other!) Test mode can be used when you need some special cases in your functions (i.e., when you are testing some functionality, which doesn't apply in normal conditions). But don't forget to turn this mode off, after you are done (within test function for example)!

5.2.1 Detailed Description

Header file that provides a data structure for the auxiliary functions

5.2.2 Function Documentation

5.2.2.1 AK_add_succesor()

```
AK_succesor AK_add_succesor (
            int succesorId,
            int succesorOf )
```

Creates an edge between two nodes.

Author

Frane Jakelić

Parameters

succesorId	id of a newly created edge
succesorOf	source of the newly created edge

Returns

pointer to the newly created edge

5.2.2.2 AK_add_vertex()

```
AK_vertex AK_add_vertex (
            int id )
```

Function that adds a new graph node.

Author

Frane Jakelić

Parameters

id	of the vertex that needs to be added
graphRoot	root node of the graph structure

Returns

pointer to the newly created node

5.2.2.3 AK_chars_num_from_number()

```
int AK_chars_num_from_number (
            int number,
            int base )
```

Function that gets the number of digits for any given number.

Author

Dino Laktašić.

Parameters

number	number to evaluate
int	base mathematic base (e.g. 2, 10 etc.)

Returns

the number of digits for the given number

5.2.2.4 AK_convert_type()

```
char* AK_convert_type (
            char * arg_type )
```

Function that change type of argument from string to integer.

Author

Aleksandra Polak

Parameters

∗*arg_type*	type of an argument

Returns

EXIT_SUCCESS of the function (return type of argument in value of integer) or EXIT_ERROR

Function that change type of argument from string to integer.

Author

Aleksandra Polak

Parameters

∗arg_type	type of argument

Returns

EXIT_SUCCESS of the function (return type of argument as a value of the integer) or EXIT_ERROR

5.2.2.5 AK_Delete_L3()

```
void AK_Delete_L3 (
            struct list_node ** current,
            struct list_node ** L )
```

Function that deletes the current element of the list.

Author

Ljiljana Pintarić.

Parameters

current	current element of the list
L	root of the list @retrun No return value

5.2.2.6 AK_DeleteAll_L3()

```
void AK_DeleteAll_L3 (
            struct list_node ** L )
```

Function that empties the list.

Author

Ljiljana Pintarić.

Parameters

L	root of the list

Returns

No return value

5.2.2.7 AK_destroy_critical_section()

```
void AK_destroy_critical_section (
            AK_synchronization_info * info )
```

Destroys a synchronization object when it is no longer necessary and frees the pointer.

Author

Marko Sinko

Parameters

info	Synchronization info structure

Returns

void

5.2.2.8 AK_End_L2()

```
struct list_node* AK_End_L2 (
            struct list_node * L )
```

Function that fetches the last element of the list.

Author

Ljiljana Pintarić.

Parameters

L	root of the list

Returns

last element of the list

5.2.2.9 AK_enter_critical_section()

```
void AK_enter_critical_section (
            AK_synchronization_info * info )
```

Enters a critical section.

Author

Marko Sinko

Parameters

info	Synchronization info structure

Returns

void

5.2.2.10 AK_First_L2()

```
struct list_node* AK_First_L2 (
            struct list_node * L )
```

Function that fetches the first element of the list.

Author

Ljiljana Pintarić.

Parameters

L	root of the list

Returns

first element of the list

5.2.2.11 AK_get_array_perms()

```
char* AK_get_array_perms (
            char * arr )
```

Get all permutations without repetition (currently not used, but it can be helpful)

Author

Dino Laktašić.

Parameters

arr	array of chars to perform permutation on

Returns

char pointer to an array of pointers pointing to permuted char arrays

Get all permutations without repetition (currently not used, but it can be helpful)

Author

Matija Novak

Parameters

SearchElement	element whose posititon we search for
L	root of the list

Returns

returns the posititon number of some elelemnt

Author

Dino Laktašić.

Get all permutations without repetition (currently not used, but it can be helpful)

Parameters

arr	array of chars to perform permutation on

Returns

char pointer to an array of pointers pointing to permuted char arrays

5.2.2.12 AK_GetNth_L2()

```
struct list_node* AK_GetNth_L2 (
            int pos,
            struct list_node * row )
```

Function that fetches the nth element in a row.

Author

Ljiljana Pintarić

Parameters

pos	position of element in a row
row	list of elements of a row in the table

Returns

element of list of elements of a row in the table

Function that fetches the nth element in a row.

Author

Matija Šestak.

Parameters

current	current list element
L	root of the list

Returns

data type of the current list element

Author

Matija Šestak.

Function that fetches the data size of the element

Parameters

current	current list element
L	- root of the list

Returns

data size of the current list element

Author

Ljiljana Pintarić

Function that fetches the nth element in a row

Parameters

pos	position of element in a row
row	list of elements of a row in the table

Returns

element of list of elements of a row in the table

5.2.2.13 AK_init_critical_section()

```
AK_synchronization_info* AK_init_critical_section ( )
```

Initializes an AK_synchronization_info structure and returns an owned pointer that must later be passed on to AK_destroy_critical_section.

Author

Marko Sinko

Returns

Initialized synchronization object

5.2.2.14 AK_Init_L3()

```
void AK_Init_L3 (
            struct list_node ** L )
```

Function that initializes an empty list.

Author

Ljiljana Pintarić

Parameters

L	root of the list

Returns

NO return value

5.2.2.15 AK_InsertAfter_L2()

```
void AK_InsertAfter_L2 (
            int type,
```

```
            char * data,
            int size,
            struct list_node ** current,
            struct list_node ** L )
```

Function that inserts a new element after the current element of the list.

Author

Ljiljana Pintarić.

Parameters

data	new data
current	current element of the list
L	root of the list

Returns

No return value.

5.2.2.16 AK_InsertAtBegin_L3()

```
void AK_InsertAtBegin_L3 (
            int type,
            char * data,
            int size,
            struct list_node * L )
```

Function that inserts a new element at the beginning of the list. It uses function called: AK_InsertBefore_L.

Author

Ljiljana Pintarić.

Parameters

data	new data
L	root of the list

Returns

No return value

5.2.2.17 AK_InsertAtEnd_L3()

```
void AK_InsertAtEnd_L3 (
            int type,
            char * data,
            int size,
            struct list_node * L )
```

Function that inserts a new element at the end of the list. It uses a function called: AK_InsertAfter_L2.

Author

Ljiljana Pintarić.

Parameters

data	new data
L	root of the list

Returns

No return value.

5.2.2.18 AK_InsertBefore_L2()

```
void AK_InsertBefore_L2 (
            int type,
            char * data,
            int size,
            struct list_node ** current,
            struct list_node ** L )
```

Function that inserts a new element before the current element of the list.

Author

Ljiljana Pintarić.

Parameters

data	new data
current	current element of the list
L	root of the list

Returns

No return value

5.2.2.19 AK_IsEmpty_L2()

```
unsigned int AK_IsEmpty_L2 (
            struct list_node * L )
```

Function that tests if the list is empty.

Author

Ljiljana Pintarić.

Parameters

L	root of the list

Returns

1 if the list is empty, otherwise returns 0

5.2.2.20 AK_leave_critical_section()

```
void AK_leave_critical_section (
            AK_synchronization_info * info )
```

Leaves a critical section.

Author

Marko Sinko

Parameters

info	Synchronization info structure

Returns

void

5.2.2.21 AK_Next_L2()

```
struct list_node* AK_Next_L2 (
            struct list_node * current )
```

Function that fetches the next element of the list.

Author

Ljiljana Pintarić.

Parameters

current	current element of the list

Returns

next element of the list

5.2.2.22 AK_pop_from_stack()

```
AK_stack AK_pop_from_stack ( )
```

Pops a entry to the stack.

Author

Frane Jakelić

Returns

pointer to the popped stack node

5.2.2.23 AK_Previous_L2()

```
struct list_node* AK_Previous_L2 (
            struct list_node * current,
            struct list_node * L )
```

Function that fetches the previous element of the list.

Author

Ljiljana Pintarić.

Parameters

current	current element of the list
L	root of the list

Returns

previous element of the list

5.2.2.24 AK_push_to_stack()

```
AK_stack AK_push_to_stack (
            int id )
```

Adds a entry to the stack.

Author

Frane Jakelić

Parameters

id	of the element that is being added to the stack

Returns

pointer to the newly added stack node

5.2.2.25 AK_Retrieve_L2()

```
char* AK_Retrieve_L2 (
            struct list_node * current,
            struct list_node * L )
```

Function that retrieves the data from the current element of the list.

Author

Ljiljana Pintarić.

Parameters

current	current element of the list
L	root of the list

Returns

data from the list element

5.2.2.26 AK_search_empty_link()

```
AK_vertex AK_search_empty_link ( )
```

Looks for empty link for a new graph node.

Author

Frane Jakelić

Parameters

graphRoot	oot node of the graph structure

Returns

empty link for a new graph node

5.2.2.27 AK_search_empty_stack_link()

```
AK_stack AK_search_empty_stack_link (
            AK_stack stackRoot )
```

Returns a empty link for the stack.

Author

Frane Jakelić

Parameters

stackRoot	root node of the selected stack

Returns

pointer to the empty link

5.2.2.28 AK_search_in_stack()

```
AK_stack AK_search_in_stack (
            int id )
```

Finds an element in the stack.

Author

Frane Jakelić

Parameters

id	of the node that needs to be found in the stack

Returns

pointer to the found stack node

5.2.2.29 AK_search_vertex()

```
AK_vertex AK_search_vertex (
            int id )
```

Function that searches for a specific graph node by its ID.

Author

Frane Jakelić

Parameters

id	of the vertex that needs to be found
graphRoot	root node of the graph structure

Returns

found graph nod or null

5.2.2.30 AK_Size_L2()

```
int AK_Size_L2 (
            struct list_node * L )
```

Function that fetches the number of the elements in the list.

Author

Ljiljana Pintarić.

Parameters

L	root of the list

Returns

Size of the list

5.2.2.31 AK_strcmp()

```
int AK_strcmp (
            const void * a,
            const void * b )
```

Function compares two Strings.

Author

Dino Laktašić

Parameters

*a	pointer of a value to compare
*b	pointer of a value to compare

Returns

result of the comparison in line with strcmp function

5.2.2.32 AK_tarjan()

```
void AK_tarjan (
            int id )
```

Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.

Author

Frane Jakelić

Parameters

id	of the element on which the algorithm looks for an id of a strongly connected component

####\nStrongy connected component. Edges:\n");

####\n");

5.2.2.33 AK_type_size()

```
size_t AK_type_size (
            int iDB_type,
            char * szVarchar )
```

Function returns the size in bytes for the provided database type.

Author

Miroslav Policki

Parameters

iDB_type	database data type (defined in constants.h)
szVarchar	if iDB_type == TYPE_VARCHAR, pointer to the string, otherwise unused

Returns

size of provided data type in bytes if the provided data type is valid, else return 0

5.2.3 Variable Documentation

5.2.3.1 testMode

`testMode`

You can turn testMode on or off with TEST_MODE_ON and TEST_MODE_OFF. To do this, simply enable or disable it in YOUR function (not in any other!) Test mode can be used when you need some special cases in your functions (i.e., when you are testing some functionality, which doesn't apply in normal conditions). But don't forget to turn this mode off, after you are done (within test function for example)!

Author

Domagoj Šitum

5.3 auxi/constants.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define MAX_VARCHAR_LENGTH 200

  Constant declaring the maximum length of varchar data value.
- #define MAX_ATTRIBUTES 10

  Constant declaring the maximum number of attributes per block.
- #define MAX_ATT_NAME 255

  Constant declaring the maximum length of attribute name string (used in AK_header->att_name)
- #define MAX_CONSTRAINTS 5

  Constant declaring the maximum number of constraints per attribute.
- #define MAX_CONSTR_NAME 255

  Constant declaring the maximum length of constraint name string (used in AK_header->constr_name)
- #define MAX_CONSTR_CODE 255

  Constant declaring the maximum lenght of constraint code string.

- #define MAX_OBSERVABLE_OBSERVERS 4096

  Constant for declaring the maximum number of observers objects for some observable type.
- #define MAX_ACTIVE_TRANSACTIONS_COUNT 100

  Constant for declaring the maximum number of active trasactions in DBMS.
- #define DATA_BLOCK_SIZE 500

  Constant declaring length of data block size (used in AK_block->data)
- #define DATA_ENTRY_SIZE 10

  Constant declaring lenght of data entry in sizeof(int)
- #define MAX_QUERY_LIB_MEMORY 255

  Constant declaring the maximum size of query lib memory.
- #define MAX_CACHE_MEMORY 255

  Constant declaring the maximum size of DB cache memory.
- #define MAX_QUERY_DICT_MEMORY 255

  Constant declaring the maximum size of query dictionary memory.
- #define MAX_QUERY_RESULT_MEMORY 255

  Constant declaring the maximum size of query result cache memory.
- #define MAX_TOKENS 255

  Constant declaring the maximum number of attributes to handle in relation equivalence function.
- #define MAX_MAIN_BUCKETS 512

  Constant declaring the maximum number of main buckets.
- #define MAIN_BUCKET_SIZE 4

  Constant declaring the size of main buckets.
- #define HASH_BUCKET_SIZE 4

  Constant declaring the size of hash buckets.
- #define NUMBER_OF_KEYS 4096

  Constant declaring the number of buckets in hash table.
- #define EXIT_SUCCESS 0

  Constant declaring a successful exit.
- #define EXIT_ERROR -1

  Constant declaring unsuccessful exit.
- #define **EXIT_WARNING** -2
- #define BLOCK_TYPE_FREE -1

  Constant declaring AK_free block type (used in AK_block->type)
- #define BLOCK_TYPE_NORMAL 0

  Constant declaring normal block type e.g. used by some extent (used in AK_block->type)
- #define BLOCK_TYPE_CHAINED 1

  Constant declaring chained block type e.g. used if the block is chained with another (used in AK_block->type)
- #define NOT_CHAINED -1

  Constant used in AK_block->chained_with if the block isn't chained.
- #define FREE_INT -10

  Constant declaring dummy data for empty integers.
- #define FREE_CHAR '\0'

  Constant declaring dummy data for empty chars.
- #define SEGMENT_TYPE_SYSTEM_TABLE 0

  Constant declaring system table segment type (used in system catalog)
- #define SEGMENT_TYPE_TABLE 1

  Constant declaring table segment type (used in system catalog)
- #define SEGMENT_TYPE_INDEX 2

  Constant declaring index segment type (used in system catalog)
- #define SEGMENT_TYPE_TRANSACTION 3

  Constant declaring transaction segment type (used in system catalog)

- #define SEGMENT_TYPE_TEMP 4

  Constant declaring temporary segment type (used in system catalog)
- #define TYPE_INTERNAL 0

  Constant declaring internal data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_INT 1

  integer data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_FLOAT 2

  Constant declaring float data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_NUMBER 3

  Constant declaring number data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_VARCHAR 4

  Constant declaring varchar data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_DATE 5

  Constant declaring date data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_DATETIME 6

  Datetime data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_TIME 7

  Constant declaring time data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_BLOB 8

  Blob data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_BOOL 9

  Constant declaring boolean data type (used in AK_header->type and AK_tuple_dict->type)
- #define TYPE_OPERAND 10

  Constant indicating operand in AK_list.
- #define TYPE_OPERATOR 11

  indicates operator in AK_list
- #define TYPE_ATTRIBS 12

  Constant indicating attribute/s in AK_list.
- #define TYPE_CONDITION 13

  Constant indicating condition in AK_list.
- #define BLOCK_CLEAN 0

  Constant indicating block cleaning (not changed since read from disk)
- #define BLOCK_DIRTY 1

  Constant indicating dirty block (changed since read from disk, has to be written)
- #define ATTR_DELIMITER ";"

  Constant declaring attributes delimiter.
- #define ATTR_ESCAPE "`"

  Constant indicating attributes escape section.
- #define NULLL "asdfgXYZ"

  Constant declaring null value for tables.
- #define **RO_SELECTION** 's'
- #define **RO_PROJECTION** 'p'
- #define **RO_NAT_JOIN** 'n'
- #define **RO_RENAME** 'r'
- #define **RO_UNION** 'u'
- #define **RO_INTERSECT** 'i'
- #define **RO_EXCEPT** 'e'
- #define **RO_THETA_JOIN** 't'
- #define NEW_VALUE 0

  Constant indicating that the data is a new value.
- #define SEARCH_CONSTRAINT 1

Constant indicating that the data is constraint to search for.

- #define UPDATE 0

  Constant indicating that the operation to be performed is 'update'.
- #define DELETE 1

  Constant indicating that the operation to be performed is 'delete'.
- #define INSERT 2

  Constant indicating that the operation to be performed is 'insert'.
- #define SELECT 3

  Constant indicating 'select' operation.
- #define FIND 2

  Constant indicating that the operation to be performed is 'search'.
- #define INFO_BUCKET 0

  Constant declaring the type of bucket as "info bucket" when inserting bucket to block.
- #define MAIN_BUCKET 1

  Constant declaring the type of bucket as "main bucket" when inserting bucket to block.
- #define HASH_BUCKET 2

  Constant declaring the type of bucket as "hash bucket" when inserting bucket to block.
- #define SHARED_LOCK 0

  Constant declaring the type of lock as SHARED LOCK.
- #define EXCLUSIVE_LOCK 1

  Constant declaring the type of lock as EXCLUSIVE LOCK.
- #define WAIT_FOR_UNLOCK 0

  Constant declaring that a lock has to wait until other locks release the resource.
- #define PASS_LOCK_QUEUE 1

  Constant declaring that a lock can acquire the resource AK_freely.
- #define OK 1

  Constant declaring that the method is completed successfuly.
- #define NOT_OK 0

  Constant declaring that the method isn't completed successfuly.
- #define COMMIT 1

  Constant declaring that the transaction is completed successfuly.
- #define ABORT 0

  Constant declaring if the transaction is being aborted.
- #define NEW_ID 0

  Constant declaring if new obj_id should be created.
- #define MAX_BLOCKS_CURRENTLY_ACCESSED 32

  Indicates the maximum number of threads that can access (read or write) database at the same time.
- #define TEST_MODE_ON 1

  This constant is used to turn testMode (auxi/auxillary.h) ON.
- #define TEST_MODE_OFF 0

  This constant is used to turn testMode (auxi/auxillary.h) OFF.
- #define SEPARATOR "[{(|&&|)}]"

  Used in unique.c for separation of names of attributes and their values when UNIQUE constraint is being set or tested on combination of values of attributes.
- #define AK_CONSTRAINTS_BEWTEEN "AK_constraints_between"

  Defines system table name for storing between constraints.
- #define AK_CONSTRAINTS_CHECK_CONSTRAINT "AK_constraints_check_constraint"

  Defines system table name for storing check constraints.
- #define AK_CONSTRAINTS_NOT_NULL "AK_constraints_not_null"

  Defines system table name for storing check constraints.
- #define AK_CONSTRAINTS_UNIQUE "AK_constraints_unique"

Defines system table name for storing check constraints.

- #define AK_CONSTRAINTS_INDEX "AK_constraints_index"

  Defines system table name for storing check constraints.
- #define AK_CONSTRAINTS_PRIMARY_KEY "AK_constraints_primary_key"

  Defines system table name for storing check constraints.
- #define AK_CONSTRAINTS_FOREIGN_KEY "AK_constraints_foreign_key"

  Defines system table name for storing check constraints.
- #define AK_CONSTRAINTS_DEFAULT "AK_constraints_default"

  Defines system table name for storing check constraints.
- #define AK_REFERENCE "AK_reference"

  Defines system table name for storing check constraints.
- #define DROP_TABLE 0

  Constant which defines the number of drop statement.
- #define DROP_INDEX 1

  Constant which defines the number of drop statement.
- #define DROP_VIEW 2

  Constant which defines the number of drop statement.
- #define DROP_SEQUENCE 3

  Constant which defines the number of drop statement.
- #define DROP_TRIGGER 4

  Constant which defines the number of drop statement.
- #define DROP_FUNCTION 5

  Constant which defines the number of drop statement.
- #define DROP_USER 6

  Constant which defines the number of drop statement.
- #define DROP_GROUP 7

  Constant which defines the number of drop statement.
- #define DROP_CONSTRAINT 8

  Constant which defines thenumber of drop statement.
- #define NUM_SYS_TABLES 20

  Constant which defines the length of system_catalog.

5.3.1 Detailed Description

Header file that provides global macros, constants and variables

5.3.2 Macro Definition Documentation

5.3.2.1 AK_CONSTRAINTS_DEFAULT

```
#define AK_CONSTRAINTS_DEFAULT "AK_constraints_default"
```

Defines system table name for storing check constraints.

- –

5.3.2.2 AK_CONSTRAINTS_FOREIGN_KEY

```
#define AK_CONSTRAINTS_FOREIGN_KEY "AK_constraints_foreign_key"
```

Defines system table name for storing check constraints.

- –

5.3.2.3 AK_CONSTRAINTS_INDEX

```
#define AK_CONSTRAINTS_INDEX "AK_constraints_index"
```

Defines system table name for storing check constraints.

- –

5.3.2.4 AK_CONSTRAINTS_PRIMARY_KEY

```
#define AK_CONSTRAINTS_PRIMARY_KEY "AK_constraints_primary_key"
```

Defines system table name for storing check constraints.

- –

5.4 auxi/debug.c File Reference

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

Functions

- int AK_dbg_messg (DEBUG_LEVEL level, DEBUG_TYPE type, const char *format,...)

  Function that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.

5.4.1 Detailed Description

Provides a function for debuging

5.4.2 Function Documentation

5.4.2.1 AK_dbg_messg()

```
int AK_dbg_messg (
            DEBUG_LEVEL level,
            DEBUG_TYPE type,
            const char * format,
             ...  )
```

Function that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.

Author

Dino Laktašić

Parameters

level	level of debug information for a given DB module
type	the name of DB module for which to print debug information
format	format for the output message
...	variable number of (different) type args used in printf

Returns

if debug message is printed return 1, else return 0

5.5 auxi/debug.h File Reference

```
#include "stdarg.h"
#include "stdio.h"
#include "stdlib.h"
#include "mempro.h"
```

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

Macros

- #define DEBUG_ALL 0

  Set constant to 1 for a complete project debug, else set constant to 0.

Typedefs

- typedef enum debug_level **DEBUG_LEVEL**
- typedef enum debug_type **DEBUG_TYPE**

Enumerations

- enum **debug_level** { **LOW** = 1, **MIDDLE** = 0, **HIGH** = 0 }
- enum **debug_type** {
  GLOBAL = 0, **DB_MAN** = 0, **FILE_MAN** = 1, **MEMO_MAN** = 0,
  INDICES = 0, **TABLES** = 0, **REL_OP** = 0, **REL_EQ** = 1,
  CONSTRAINTS = 0, **FUNCTIONS** = 0, **SEQUENCES** = 0, **TRIGGERS** = 0,
  REDO = 0 }

Functions

- int AK_dbg_messg (DEBUG_LEVEL level, DEBUG_TYPE type, const char *format,...)

  Function that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.

5.5.1 Detailed Description

Header file that defines global macros, constants and variables for debuging

5.5.2 Macro Definition Documentation

5.5.2.1 DEBUG_ALL

```
#define DEBUG_ALL 0
```

Set constant to 1 for a complete project debug, else set constant to 0.

Author

Dino Laktašić

5.5.3 Function Documentation

5.5.3.1 AK_dbg_messg()

```
int AK_dbg_messg (
            DEBUG_LEVEL level,
            DEBUG_TYPE type,
            const char * format,
             ...  )
```

Function that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.

Author

Dino Laktašić

Parameters

level	level of debug information for a given DB module
type	the name of DB module for which to print debug information
format	format for the output message
...	variable number of (different) type args used in printf

Returns

if debug message is printed return 1, else return 0

5.6 auxi/dictionary.c File Reference

Implements a dictionary for string variables.

```
#include "dictionary.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for dictionary.c:

Macros

- #define MAXVALSZ 1024
- #define DICTMINSZ 128
- #define DICT_INVALID_KEY ((char∗)-1)

Functions

- unsigned dictionary_hash (const char ∗key)

  Compute the hash key for a string.
- dictionary ∗ dictionary_new (int size)

  Create a new dictionary object.
- void dictionary_del (dictionary ∗d)

  Delete a dictionary object.
- char ∗ dictionary_get (dictionary ∗d, const char ∗key, char ∗def)

  Get a value from a dictionary.
- int dictionary_set (dictionary ∗d, const char ∗key, const char ∗val)

  Set a value in a dictionary.
- void dictionary_unset (dictionary ∗d, const char ∗key)

  Delete a key in a dictionary.
- void dictionary_dump (dictionary ∗d, FILE ∗out)

  Dump a dictionary to an opened file pointer.

5.6.1 Detailed Description

Implements a dictionary for string variables.

Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

5.6.2 Macro Definition Documentation

5.6.2.1 DICT_INVALID_KEY

```
#define DICT_INVALID_KEY ((char*)-1)
```

Invalid key token

5.6.2.2 DICTMINSZ

```
#define DICTMINSZ 128
```

Minimal allocated number of entries in a dictionary

5.6.2.3 MAXVALSZ

```
#define MAXVALSZ 1024
```

Maximum value size for integers and doubles.

5.6.3 Function Documentation

5.6.3.1 dictionary_del()

```
void dictionary_del (
            dictionary * d )
```

Delete a dictionary object.

Parameters

d	dictionary object to deallocate.

Returns

void

Deallocate a dictionary object and all memory associated to it.

5.6.3.2 dictionary_dump()

```
void dictionary_dump (
            dictionary * d,
            FILE * out )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump
f	Opened file pointer.

Returns

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

5.6.3.3 dictionary_get()

```
char* dictionary_get (
            dictionary * d,
            const char * key,
            char * def )
```

Get a value from a dictionary.

Parameters

d	dictionary object to search.
key	Key to look for in the dictionary.
def	Default value to return if key not found.

Returns

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to AK_free it or modify it.

5.6.3.4 dictionary_hash()

```
unsigned dictionary_hash (
            const char * key )
```

Compute the hash key for a string.

Parameters

key	Character string to use for key.

Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-AK_free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

5.6.3.5 dictionary_new()

```
dictionary* dictionary_new (
            int size )
```

Create a new dictionary object.

Parameters

size	Optional initial size of the dictionary.

Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

5.6.3.6 dictionary_set()

```
int dictionary_set (
            dictionary * d,
            const char * key,
            const char * val )
```

Set a value in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to modify or add.
val	Value to add.

Returns

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary_set a variable to NULL, a call to dictionary_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

5.6.3.7 dictionary_unset()

```
void dictionary_unset (
            dictionary * d,
            const char * key )
```

Delete a key in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to remove.

Returns

void

This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

5.7 auxi/dictionary.h File Reference

Implements a dictionary for string variables.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "mempro.h"
```

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

Classes

- struct _dictionary_

  Dictionary object.

Typedefs

- typedef struct _dictionary_ dictionary
  Dictionary object.

Functions

- unsigned dictionary_hash (const char *key)
  Compute the hash key for a string.
- dictionary * dictionary_new (int size)
  Create a new dictionary object.
- void dictionary_del (dictionary *vd)
  Delete a dictionary object.
- char * dictionary_get (dictionary *d, const char *key, char *def)
  Get a value from a dictionary.
- int dictionary_set (dictionary *vd, const char *key, const char *val)
  Set a value in a dictionary.
- void dictionary_unset (dictionary *d, const char *key)
  Delete a key in a dictionary.
- void dictionary_dump (dictionary *d, FILE *out)
  Dump a dictionary to an opened file pointer.

5.7.1 Detailed Description

Implements a dictionary for string variables.

Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

5.7.2 Typedef Documentation

5.7.2.1 dictionary

```
typedef struct _dictionary_ dictionary
```

Dictionary object.

This object contains a list of string/string associations. Each association is identified by a unique string key. Looking up values in the dictionary is speeded up by the use of a (hopefully collision-AK_free) hash function.

5.7.3 Function Documentation

5.7.3.1 dictionary_del()

```
void dictionary_del (
            dictionary * d )
```

Delete a dictionary object.

Parameters

d	dictionary object to deallocate.

Returns

void

Deallocate a dictionary object and all memory associated to it.

5.7.3.2 dictionary_dump()

```
void dictionary_dump (
            dictionary * d,
            FILE * out )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump
f	Opened file pointer.

Returns

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

5.7.3.3 dictionary_get()

```
char* dictionary_get (
            dictionary * d,
            const char * key,
            char * def )
```

Get a value from a dictionary.

Parameters

d	dictionary object to search.
key	Key to look for in the dictionary.
def	Default value to return if key not found.

Returns

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to AK_free it or modify it.

5.7.3.4 dictionary_hash()

```
unsigned dictionary_hash (
            const char * key )
```

Compute the hash key for a string.

Parameters

key	Character string to use for key.

Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-AK_free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

5.7.3.5 dictionary_new()

```
dictionary* dictionary_new (
            int size )
```

Create a new dictionary object.

Parameters

size	Optional initial size of the dictionary.

Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

5.7.3.6 dictionary_set()

```
int dictionary_set (
            dictionary * d,
            const char * key,
            const char * val )
```

Set a value in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to modify or add.
val	Value to add.

Returns

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary_set a variable to NULL, a call to dictionary_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

5.7.3.7 dictionary_unset()

```
void dictionary_unset (
            dictionary * d,
            const char * key )
```

Delete a key in a dictionary.

Parameters

d	dictionary object to modify.
key	Key to remove.

Returns

void

This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

5.8 auxi/iniparser.c File Reference

Parser for ini files.

```
#include <ctype.h>
#include "iniparser.h"
```

Include dependency graph for iniparser.c:

Macros

- #define **ASCIILINESZ** (1024)
- #define **INI_INVALID_KEY** ((char∗)-1)

Typedefs

- typedef enum _line_status_ line_status

Enumerations

- enum _line_status_ {
  LINE_UNPROCESSED, **LINE_ERROR**, **LINE_EMPTY**, **LINE_COMMENT**,
  LINE_SECTION, **LINE_VALUE** }

Functions

- int iniparser_getnsec (dictionary ∗d)

  Get number of sections in a dictionary.
- char ∗ iniparser_getsecname (dictionary ∗d, int n)

  Get name for section n in a dictionary.
- void iniparser_dump (dictionary ∗d, FILE ∗f)

  Dump a dictionary to an opened file pointer.
- void iniparser_dump_ini (dictionary ∗d, FILE ∗f)

  Save a dictionary to a loadable ini file.
- void iniparser_dumpsection_ini (dictionary ∗d, char ∗s, FILE ∗f)

  Save a dictionary section to a loadable ini file.
- int iniparser_getsecnkeys (dictionary ∗d, char ∗s)

  Get the number of keys in a section of a dictionary.
- char ∗∗ iniparser_getseckeys (dictionary ∗d, char ∗s)

  Get the number of keys in a section of a dictionary.
- char ∗ iniparser_getstring (dictionary ∗d, const char ∗key, char ∗def)

  Get the string associated to a key.
- int iniparser_getint (dictionary ∗d, const char ∗key, int notfound)

  Get the string associated to a key, convert to an int.
- double iniparser_getdouble (dictionary ∗d, const char ∗key, double notfound)

  Get the string associated to a key, convert to a double.
- int iniparser_getboolean (dictionary ∗d, const char ∗key, int notfound)

  Get the string associated to a key, convert to a boolean.
- int iniparser_find_entry (dictionary ∗ini, const char ∗entry)

  Finds out if a given entry exists in a dictionary.
- int iniparser_set (dictionary ∗ini, const char ∗entry, const char ∗val)

  Set an entry in a dictionary.
- void iniparser_unset (dictionary ∗ini, const char ∗entry)

  Delete an entry in a dictionary.
- dictionary ∗ iniparser_load (const char ∗ininame)

  Parse an ini file and return an allocated dictionary object.
- void iniparser_AK_freedict (dictionary ∗d)

  Free all memory associated to an ini dictionary.
- void **AK_inflate_config** ()

Variables

- pthread_mutex_t **iniParserMutex** = PTHREAD_MUTEX_INITIALIZER
- dictionary * **AK_config**

5.8.1 Detailed Description

Parser for ini files.

Author

N. Devillard

5.8.2 Typedef Documentation

5.8.2.1 line_status

```
typedef enum _line_status_ line_status
```

This enum stores the status for each parsed line (internal use only).

5.8.3 Enumeration Type Documentation

5.8.3.1 _line_status_

```
enum _line_status_
```

This enum stores the status for each parsed line (internal use only).

5.8.4 Function Documentation

5.8.4.1 iniparser_AK_freedict()

```
void iniparser_AK_freedict (
            dictionary * d )
```

Free all memory associated to an ini dictionary.

Parameters

d	Dictionary to AK_free

Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

5.8.4.2 iniparser_dump()

```
void iniparser_dump (
             dictionary * d,
             FILE * f )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump.
f	Opened file pointer to dump to.

Returns

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

5.8.4.3 iniparser_dump_ini()

```
void iniparser_dump_ini (
             dictionary * d,
             FILE * f )
```

Save a dictionary to a loadable ini file.

Parameters

d	Dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

5.8.4.4 iniparser_dumpsection_ini()

```
void iniparser_dumpsection_ini (
            dictionary * d,
            char * s,
            FILE * f )
```

Save a dictionary section to a loadable ini file.

Parameters

d	Dictionary to dump
s	Section name of dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

5.8.4.5 iniparser_find_entry()

```
int iniparser_find_entry (
            dictionary * ini,
            const char * entry )
```

Finds out if a given entry exists in a dictionary.

Parameters

ini	Dictionary to search
entry	Name of the entry to look for

Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

5.8.4.6 iniparser_getboolean()

```
int iniparser_getboolean (
            dictionary * d,
```

```
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to a boolean.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'
- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

5.8.4.7 iniparser_getdouble()

```
double iniparser_getdouble (
            dictionary * d,
            const char * key,
            double notfound )
```

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

5.8.4.8 iniparser_getint()

```
int iniparser_getint (
            dictionary * d,
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to an int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

5.8.4.9 iniparser_getnsec()

```
int iniparser_getnsec (
            dictionary * d )
```

Get number of sections in a dictionary.

Parameters

d	Dictionary to examine

Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

5.8.4.10 iniparser_getseckeys()

```
char** iniparser_getseckeys (
            dictionary * d,
            char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

pointer to statically allocated character strings

This function queries a dictionary and finds all keys in a given section. Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not AK_free or modify them.

This function returns NULL in case of error.

5.8.4.11 iniparser_getsecname()

```
char* iniparser_getsecname (
            dictionary * d,
            int n )
```

Get name for section n in a dictionary.

Parameters

d	Dictionary to examine
n	Section number (from 0 to nsec-1).

Returns

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not AK_free or modify the returned string!

This function returns NULL in case of error.

5.8.4.12 iniparser_getsecnkeys()

```
int iniparser_getsecnkeys (
            dictionary * d,
            char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

Number of keys in section

5.8.4.13 iniparser_getstring()

```
char* iniparser_getstring (
            dictionary * d,
            const char * key,
            char * def )
```

Get the string associated to a key.

Parameters

d	Dictionary to search
key	Key string to look for
def	Default value to return if key not found.

Returns

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not AK_free or modify it.

5.8.4.14 iniparser_load()

```
dictionary* iniparser_load (
            const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

Parameters

ininame	Name of the ini file to read.

Returns

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be AK_freed using iniparser_AK_freedict().

5.8.4.15 iniparser_set()

```
int iniparser_set (
            dictionary * ini,
            const char * entry,
            const char * val )
```

Set an entry in a dictionary.

Parameters

ini	Dictionary to modify.
entry	Entry to modify (entry name)
val	New value to associate to the entry.

Returns

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, -1 is returned. It is Ok to set val to NULL.

5.8.4.16 iniparser_unset()

```
void iniparser_unset (
            dictionary * ini,
            const char * entry )
```

Delete an entry in a dictionary.

Parameters

ini	Dictionary to modify
entry	Entry to delete (entry name)

Returns

void

If the given entry can be found, it is deleted from the dictionary.

5.9 auxi/iniparser.h File Reference

Parser for ini files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "dictionary.h"
#include "mempro.h"
```

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

Functions

- int iniparser_getnsec (dictionary ∗d)
  Get number of sections in a dictionary.
- char ∗ iniparser_getsecname (dictionary ∗d, int n)
  Get name for section n in a dictionary.
- void iniparser_dump_ini (dictionary ∗d, FILE ∗f)
  Save a dictionary to a loadable ini file.
- void iniparser_dumpsection_ini (dictionary ∗d, char ∗s, FILE ∗f)
  Save a dictionary section to a loadable ini file.
- void iniparser_dump (dictionary ∗d, FILE ∗f)
  Dump a dictionary to an opened file pointer.
- int iniparser_getsecnkeys (dictionary ∗d, char ∗s)
  Get the number of keys in a section of a dictionary.
- char ∗∗ iniparser_getseckeys (dictionary ∗d, char ∗s)
  Get the number of keys in a section of a dictionary.
- char ∗ iniparser_getstring (dictionary ∗d, const char ∗key, char ∗def)
  Get the string associated to a key.
- int iniparser_getint (dictionary ∗d, const char ∗key, int notfound)
  Get the string associated to a key, convert to an int.
- double iniparser_getdouble (dictionary ∗d, const char ∗key, double notfound)
  Get the string associated to a key, convert to a double.
- int iniparser_getboolean (dictionary ∗d, const char ∗key, int notfound)
  Get the string associated to a key, convert to a boolean.
- int iniparser_set (dictionary ∗ini, const char ∗entry, const char ∗val)
  Set an entry in a dictionary.

- void iniparser_unset (dictionary *ini, const char *entry)

  Delete an entry in a dictionary.
- int iniparser_find_entry (dictionary *ini, const char *entry)

  Finds out if a given entry exists in a dictionary.
- dictionary * iniparser_load (const char *ininame)

  Parse an ini file and return an allocated dictionary object.
- void iniparser_AK_freedict (dictionary *d)

  Free all memory associated to an ini dictionary.
- void **AK_inflate_config** ()

Variables

- dictionary * **AK_config**

5.9.1 Detailed Description

Parser for ini files.

Author

N. Devillard

5.9.2 Function Documentation

5.9.2.1 iniparser_AK_freedict()

```
void iniparser_AK_freedict (
            dictionary * d )
```

Free all memory associated to an ini dictionary.

Parameters

d	Dictionary to AK_free

Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

5.9.2.2 iniparser_dump()

```
void iniparser_dump (
            dictionary * d,
            FILE * f )
```

Dump a dictionary to an opened file pointer.

Parameters

d	Dictionary to dump.
f	Opened file pointer to dump to.

Returns

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

5.9.2.3 iniparser_dump_ini()

```
void iniparser_dump_ini (
            dictionary * d,
            FILE * f )
```

Save a dictionary to a loadable ini file.

Parameters

d	Dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

5.9.2.4 iniparser_dumpsection_ini()

```
void iniparser_dumpsection_ini (
            dictionary * d,
            char * s,
            FILE * f )
```

Save a dictionary section to a loadable ini file.

Parameters

d	Dictionary to dump
s	Section name of dictionary to dump
f	Opened file pointer to dump to

Returns

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

5.9.2.5 iniparser_find_entry()

```
int iniparser_find_entry (
            dictionary * ini,
            const char * entry )
```

Finds out if a given entry exists in a dictionary.

Parameters

ini	Dictionary to search
entry	Name of the entry to look for

Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

5.9.2.6 iniparser_getboolean()

```
int iniparser_getboolean (
            dictionary * d,
            const char * key,
            int notfound )
```

Get the string associated to a key, convert to a boolean.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'
- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

5.9.2.7 iniparser_getdouble()

```
double iniparser_getdouble (
             dictionary * d,
             const char * key,
             double notfound )
```

Get the string associated to a key, convert to a double.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

5.9.2.8 iniparser_getint()

```
int iniparser_getint (
             dictionary * d,
             const char * key,
             int notfound )
```

Get the string associated to a key, convert to an int.

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

- "42" -> 42
- "042" -> 34 (octal -> decimal)
- "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

Parameters

d	Dictionary to search
key	Key string to look for
notfound	Value to return in case of error

Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

5.9.2.9 iniparser_getnsec()

```
int iniparser_getnsec (
            dictionary * d )
```

Get number of sections in a dictionary.

Parameters

d	Dictionary to examine

Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

5.9.2.10 iniparser_getseckeys()

```
char** iniparser_getseckeys (
            dictionary * d,
            char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

pointer to statically allocated character strings

This function queries a dictionary and finds all keys in a given section. Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not AK_free or modify them.

This function returns NULL in case of error.

5.9.2.11 iniparser_getsecname()

```
char* iniparser_getsecname (
            dictionary * d,
            int n )
```

Get name for section n in a dictionary.

Parameters

d	Dictionary to examine
n	Section number (from 0 to nsec-1).

Returns

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not AK_free or modify the returned string!

This function returns NULL in case of error.

5.9.2.12 iniparser_getsecnkeys()

```
int iniparser_getsecnkeys (
            dictionary * d,
            char * s )
```

Get the number of keys in a section of a dictionary.

Parameters

d	Dictionary to examine
s	Section name of dictionary to examine

Returns

Number of keys in section

5.9.2.13 iniparser_getstring()

```
char* iniparser_getstring (
            dictionary * d,
            const char * key,
            char * def )
```

Get the string associated to a key.

Parameters

d	Dictionary to search
key	Key string to look for
def	Default value to return if key not found.

Returns

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not AK_free or modify it.

5.9.2.14 iniparser_load()

```
dictionary* iniparser_load (
            const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

Parameters

ininame	Name of the ini file to read.

Returns

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be AK_freed using iniparser_AK_freedict().

5.9.2.15 iniparser_set()

```
int iniparser_set (
            dictionary * ini,
            const char * entry,
            const char * val )
```

Set an entry in a dictionary.

Parameters

ini	Dictionary to modify.
entry	Entry to modify (entry name)
val	New value to associate to the entry.

Returns

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, -1 is returned. It is Ok to set val to NULL.

5.9.2.16 iniparser_unset()

```
void iniparser_unset (
            dictionary * ini,
            const char * entry )
```

Delete an entry in a dictionary.

Parameters

ini	Dictionary to modify
entry	Entry to delete (entry name)

Returns

void

If the given entry can be found, it is deleted from the dictionary.

5.10 auxi/mempro.c File Reference

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

Functions

- void AK_debmod_d (AK_debmod_state *ds, const char *message)
  Function prints debug message [private function].
- void AK_debmod_dv (AK_debmod_state *ds, const char *format,...)
  Function prints debug message [private function].
- void AK_debmod_enter_critical_sec (AK_debmod_state *ds)
  Reserves ds for use [private function].
- void AK_debmod_leave_critical_sec (AK_debmod_state *ds)
  Makes ds available [private function].
- AK_debmod_state * AK_debmod_init (void)
  Initializes debug mode structure [public function].
- void AK_debmod_die (AK_debmod_state *ds)
  Destroy debug mode state (call before main() exit) [public function].
- void * AK_debmod_calloc (AK_debmod_state *ds, uint32_t size)
  Allocates memory [private function].
- void AK_debmod_free (AK_debmod_state *ds, void *memory)
  Frees memory allocated with debmod_alloc [private function].
- void * AK_calloc (size_t num, size_t size)
  Allocates memory (see calloc) [public function].
- void * AK_malloc (size_t size)
  Allocate memory (see malloc) [public function].
- void AK_free (void *ptr)
  Free memory at ptr (see free) [public function].
- void * AK_realloc (void *ptr, size_t size)
  Reallocates memory (see realloc) [public function].
- void AK_write_protect (void *memory)
  Function write-protects memory [public function].
- void AK_write_unprotect (void *memory)
  Function write-unprotects memory [public function].
- void AK_check_for_writes (void)
  Marks pages dirty if there were writes between calls to this function.

- int32_t AK_debmod_func_id (AK_debmod_state *ds, const char *func_name)
  Returns function id for given func_name.
- const char * AK_debmod_func_get_name (AK_debmod_state *ds, int32_t function_id)
  Lookup function name [private function].
- int32_t AK_debmod_func_add (AK_debmod_state *ds, const char *func_name)
  Adds function name to list [private function].
- void AK_debmod_fstack_push (AK_debmod_state *ds, int32_t func_id)
  Push function id on stack [private function].
- int32_t AK_debmod_fstack_pop (AK_debmod_state *ds)
  Pops function id from stack [private function].
- void AK_debmod_function_current (AK_debmod_state *ds, int32_t new_function_id)
  Sets current function [private function].
- void AK_debmod_function_prologue (const char *func_name, const char *source_file, int source_line)
  Not for direct use (only with macro AK_PRO). Marks function prologue.
- void AK_debmod_log_memory_alloc (int32_t func_id)
  print debmod information on function [private function]
- void AK_debmod_function_epilogue (const char *func_name, const char *source_file, int source_line)
  Not for direct use (only with macro AK_EPI). Marks function epilogue.
- void AK_debmod_print_function_use (const char *func_name, uint8_t in_recur)
  Print function dependency [private function].
- void AK_print_function_use (const char *func_name)
  Print function dependency [public function].
- void AK_print_function_uses ()
  Print function dependency for all functions [public function].
- void AK_print_active_functions ()
  Print all detected functions.
- size_t AK_fwrite (const void *buf, size_t size, size_t count, FILE *fp)
  Write to a file from a buffer (see fwrite) [public function].
- size_t AK_fread (void *buf, size_t size, size_t count, FILE *fp)
  Read from a file (see fread) [public function].
- void AK_mempro_test ()
  Test function.

5.10.1 Detailed Description

Implementation of the memory wrappers and debug mode of Kalashnikov DB.

5.10.2 Function Documentation

5.10.2.1 AK_calloc()

```
void* AK_calloc (
            size_t num,
            size_t size )
```

Allocates memory (see calloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

num	number of elements
size	of element in bytes

Returns

allocated memory or NULL

5.10.2.2 AK_check_for_writes()

```
void AK_check_for_writes (
            void  )
```

Marks pages dirty if there were writes between calls to this function.

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.10.2.3 AK_debmod_calloc()

```
void* AK_debmod_calloc (
            AK_debmod_state * ds,
            uint32_t size )
```

Allocates memory [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
size	in bytes to allocate

Returns

pointer to allocated memory or NULL

5.10.2.4 AK_debmod_d()

```
void AK_debmod_d (
            AK_debmod_state * ds,
            const char * message )
```

Function prints debug message [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
message	string to print

Returns

void

5.10.2.5 AK_debmod_die()

```
void AK_debmod_die (
            AK_debmod_state * ds )
```

Destroy debug mode state (call before main() exit) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.10.2.6 AK_debmod_dv()

```
void AK_debmod_dv (
            AK_debmod_state * ds,
            const char * format,
             ...  )
```

Function prints debug message [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
format	format string like printf

Returns

void

5.10.2.7 AK_debmod_enter_critical_sec()

```
void AK_debmod_enter_critical_sec (
            AK_debmod_state * ds )
```

Reserves ds for use [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.10.2.8 AK_debmod_free()

```
void AK_debmod_free (
            AK_debmod_state * ds,
            void * memory )
```

Frees memory allocated with debmod_alloc [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
memory	

Returns

void

5.10.2.9 AK_debmod_fstack_pop()

```
int32_t AK_debmod_fstack_pop (
            AK_debmod_state * ds )
```

Pops function id from stack [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

function id popped

5.10.2.10 AK_debmod_fstack_push()

```
void AK_debmod_fstack_push (
            AK_debmod_state * ds,
            int32_t func_id )
```

Push function id on stack [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func↩ _id	function id

Returns

void

5.10.2.11 AK_debmod_func_add()

```
int32_t AK_debmod_func_add (
            AK_debmod_state * ds,
            const char * func_name )
```

Adds function name to list [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func_name	

Returns

id for added function name

5.10.2.12 AK_debmod_func_get_name()

```
const char* AK_debmod_func_get_name (
            AK_debmod_state * ds,
            int32_t function_id )
```

Lookup function name [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
function↩ _id	

Returns

function name for given function_id

5.10.2.13 AK_debmod_func_id()

```
int32_t AK_debmod_func_id (
            AK_debmod_state * ds,
            const char * func_name )
```

Returns function id for given func_name.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func_name	function name [private function]

Returns

function id

5.10.2.14 AK_debmod_function_current()

```
void AK_debmod_function_current (
            AK_debmod_state * ds,
            int32_t new_function_id )
```

Sets current function [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
new_function↩ _id	

Returns

void

5.10.2.15 AK_debmod_function_epilogue()

```
void AK_debmod_function_epilogue (
            const char * func_name,
            const char * source_file,
            int source_line )
```

Not for direct use (only with macro AK_EPI). Marks function epilogue.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name as in source
source_file	file name where function is defined
source_line	line from which this function is called

Returns

void

5.10.2.16 AK_debmod_function_prologue()

```
void AK_debmod_function_prologue (
            const char * func_name,
            const char * source_file,
            int source_line )
```

Not for direct use (only with macro AK_PRO). Marks function prologue.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name as in source
source_file	file name where function is defined
source_line	line from which this function is called

Returns

void

5.10.2.17 AK_debmod_init()

```
AK_debmod_state* AK_debmod_init (
            void  )
```

Initializes debug mode structure [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

initialized debug mode state

5.10.2.18 AK_debmod_leave_critical_sec()

```
void AK_debmod_leave_critical_sec (
            AK_debmod_state * ds )
```

Makes ds available [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.10.2.19 AK_debmod_log_memory_alloc()

```
void AK_debmod_log_memory_alloc (
            int32_t func_id )
```

print debmod information on function [private function]

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_id	calling function id

Returns

void

5.10.2.20 AK_debmod_print_function_use()

```
void AK_debmod_print_function_use (
            const char * func_name,
            uint8_t in_recur )
```

Print function dependency [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name
in_recur	called in recursion

Returns

void

5.10.2.21 AK_fread()

```
size_t AK_fread (
            void * buf,
```

```
            size_t size,
            size_t count,
            FILE * fp )
```

Read from a file (see fread) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

number of items read

5.10.2.22 AK_free()

```
void AK_free (
            void * ptr )
```

Free memory at ptr (see free) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ptr	pointer to memory

Returns

void

5.10.2.23 AK_fwrite()

```
size_t AK_fwrite (
            const void * buf,
            size_t size,
            size_t count,
            FILE * fp )
```

Write to a file from a buffer (see fwrite) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

number of items written

5.10.2.24 AK_malloc()

```
void* AK_malloc (
            size_t size )
```

Allocate memory (see malloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

size	of memory to allocate in bytes

Returns

allocated memory or NULL

5.10.2.25 AK_mempro_test()

```
void AK_mempro_test ( )
```

Test function.

Author

Ivan Kristo

5.10.2.26 AK_print_active_functions()

```
void AK_print_active_functions ( )
```

Print all detected functions.

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.10.2.27 AK_print_function_use()

```
void AK_print_function_use (
            const char * func_name )
```

Print function dependency [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name

Returns

void

5.10.2.28 AK_print_function_uses()

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.10.2.29 AK_realloc()

```
void* AK_realloc (
            void * ptr,
            size_t size )
```

Reallocates memory (see realloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ptr	old memory
size	new size

Returns

reallocated memory or NULL

5.10.2.30 AK_write_protect()

```
void AK_write_protect (
            void * memory )
```

Function write-protects memory [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

memory	

Returns

void

5.10.2.31 AK_write_unprotect()

```
void AK_write_unprotect (
            void * memory )
```

Function write-unprotects memory [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

memory	

Returns

void

5.11 auxi/mempro.h File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include <stdarg.h>
```

Include dependency graph for mempro.h:

Classes

- struct AK_debmod_state

  Global structure that holds all relevant information for the debug mode and related functionality.

Macros

- #define **NEW**(type, type_size) (calloc(type_size, sizeof(type)))
- #define **AK_INLINE** __inline__
- #define AK_DEBMOD_ON 0

  Zero to switch memory protection and debug mode off.
- #define AK_DEBMOD_PRINT 0

  Defines if the debug mode messages are going to be printed.
- #define AK_DEBMOD_PAGES_NUM 8192

  Defines the total available memory pages for allocation.
- #define AK_DEBMOD_MAX_WRITE_DETECTIONS (AK_DEBMOD_PAGES_NUM * 10)

  Defines the maximum number of memory write detections.
- #define AK_DEBMOD_STACKSIZE AK_DEBMOD_PAGES_NUM

  Defines the monitored functions stack.
- #define AK_DEBMOD_MAX_FUNCTIONS 500

  Defines the maximum number of function names in the application.
- #define AK_DEBMOD_MAX_FUNC_NAME 80

  Defines the maximum function name length possible.
- #define AK_PRO AK_debmod_function_prologue(__func__, __FILE__, __LINE__);

  Mandatory function prologue for all functions (AK_debmod and related functions are excluded). Put this macro after variable declarations, before any function instruction.
- #define AK_EPI AK_debmod_function_epilogue(__func__, __FILE__, __LINE__);

  Mandatory function epilogue for all functions (AK_debmod and related functions are excluded). Put this macro after last function instruction, before every return statement.

Functions

- void AK_debmod_d (AK_debmod_state *, const char *)

  Function prints debug message [private function].
- void AK_debmod_dv (AK_debmod_state *, const char *,...)

  Function prints debug message [private function].
- void AK_debmod_enter_critical_sec (AK_debmod_state *)

  Reserves ds for use [private function].
- void AK_debmod_leave_critical_sec (AK_debmod_state *)

  Makes ds available [private function].
- AK_debmod_state * AK_debmod_init (void)

  Initializes debug mode structure [public function].
- void AK_debmod_die (AK_debmod_state *)

  Destroy debug mode state (call before main() exit) [public function].
- void * AK_debmod_calloc (AK_debmod_state *, uint32_t)

  Allocates memory [private function].
- void AK_debmod_free (AK_debmod_state *, void *)

  Frees memory allocated with debmod_alloc [private function].
- void * AK_calloc (size_t, size_t)

  Allocates memory (see calloc) [public function].
- void * AK_malloc (size_t)

  Allocate memory (see malloc) [public function].
- void AK_free (void *)

  Free memory at ptr (see free) [public function].
- void * AK_realloc (void *, size_t)

  Reallocates memory (see realloc) [public function].
- void AK_write_protect (void *)

  Function write-protects memory [public function].
- void AK_write_unprotect (void *)

  Function write-unprotects memory [public function].
- void AK_check_for_writes (void)

  Marks pages dirty if there were writes between calls to this function.
- int32_t AK_debmod_func_id (AK_debmod_state *, const char *)

  Returns function id for given func_name.
- const char * AK_debmod_func_get_name (AK_debmod_state *, int32_t)

  Lookup function name [private function].
- int32_t AK_debmod_func_add (AK_debmod_state *, const char *)

  Adds function name to list [private function].
- void AK_debmod_fstack_push (AK_debmod_state *, int32_t)

  Push function id on stack [private function].
- int32_t AK_debmod_fstack_pop (AK_debmod_state *)

  Pops function id from stack [private function].
- void AK_debmod_function_current (AK_debmod_state *, int32_t)

  Sets current function [private function].
- void AK_debmod_function_prologue (const char *, const char *, int)

  Not for direct use (only with macro AK_PRO). Marks function prologue.
- void AK_debmod_function_epilogue (const char *, const char *, int)

  Not for direct use (only with macro AK_EPI). Marks function epilogue.
- void AK_debmod_log_memory_alloc (int32_t)

  print debmod information on function [private function]
- void AK_debmod_print_function_use (const char *, uint8_t)

Print function dependency [private function].

- void AK_print_function_use (const char *)

  Print function dependency [public function].

- void AK_print_function_uses ()

  Print function dependency for all functions [public function].

- void AK_print_active_functions ()

  Print all detected functions.

- void AK_mempro_test ()

  Test function.

Variables

- AK_debmod_state * **AK_DEBMOD_STATE**

5.11.1 Detailed Description

Data structures, includes, macros and declarations for the memory wrappers and debug mode of Kalashnikov DB.

5.11.2 Function Documentation

5.11.2.1 AK_calloc()

```
void* AK_calloc (
            size_t num,
            size_t size )
```

Allocates memory (see calloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

num	number of elements
size	of element in bytes

Returns

allocated memory or NULL

5.11.2.2 AK_check_for_writes()

```
void AK_check_for_writes (
            void  )
```

Marks pages dirty if there were writes between calls to this function.

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.11.2.3 AK_debmod_calloc()

```
void* AK_debmod_calloc (
            AK_debmod_state * ds,
            uint32_t size )
```

Allocates memory [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
size	in bytes to allocate

Returns

pointer to allocated memory or NULL

5.11.2.4 AK_debmod_d()

```
void AK_debmod_d (
            AK_debmod_state * ds,
            const char * message )
```

Function prints debug message [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
message	string to print

Returns

void

5.11.2.5 AK_debmod_die()

```
void AK_debmod_die (
            AK_debmod_state * ds )
```

Destroy debug mode state (call before main() exit) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.11.2.6 AK_debmod_dv()

```
void AK_debmod_dv (
            AK_debmod_state * ds,
            const char * format,
             ...  )
```

Function prints debug message [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
format	format string like printf

Returns

void

5.11.2.7 AK_debmod_enter_critical_sec()

```
void AK_debmod_enter_critical_sec (
            AK_debmod_state * ds )
```

Reserves ds for use [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.11.2.8 AK_debmod_free()

```
void AK_debmod_free (
            AK_debmod_state * ds,
            void * memory )
```

Frees memory allocated with debmod_alloc [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
memory	

Returns

void

5.11.2.9 AK_debmod_fstack_pop()

```
int32_t AK_debmod_fstack_pop (
            AK_debmod_state * ds )
```

Pops function id from stack [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

function id popped

5.11.2.10 AK_debmod_fstack_push()

```
void AK_debmod_fstack_push (
            AK_debmod_state * ds,
            int32_t func_id )
```

Push function id on stack [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func_id	function id

Returns

void

5.11.2.11 AK_debmod_func_add()

```
int32_t AK_debmod_func_add (
            AK_debmod_state * ds,
            const char * func_name )
```

Adds function name to list [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func_name	

Returns

id for added function name

5.11.2.12 AK_debmod_func_get_name()

```
const char* AK_debmod_func_get_name (
            AK_debmod_state * ds,
            int32_t function_id )
```

Lookup function name [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
function↩ _id	

Returns

function name for given function_id

5.11.2.13 AK_debmod_func_id()

```
int32_t AK_debmod_func_id (
            AK_debmod_state * ds,
            const char * func_name )
```

Returns function id for given func_name.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
func_name	function name [private function]

Returns

function id

5.11.2.14 AK_debmod_function_current()

```
void AK_debmod_function_current (
            AK_debmod_state * ds,
            int32_t new_function_id )
```

Sets current function [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state
new_function↩ _id	

Returns

void

5.11.2.15 AK_debmod_function_epilogue()

```
void AK_debmod_function_epilogue (
            const char * func_name,
            const char * source_file,
            int source_line )
```

Not for direct use (only with macro AK_EPI). Marks function epilogue.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name as in source
source_file	file name where function is defined
source_line	line from which this function is called

Returns

void

5.11.2.16 AK_debmod_function_prologue()

```
void AK_debmod_function_prologue (
            const char * func_name,
            const char * source_file,
            int source_line )
```

Not for direct use (only with macro AK_PRO). Marks function prologue.

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name as in source
source_file	file name where function is defined
source_line	line from which this function is called

Returns

void

5.11.2.17 AK_debmod_init()

```
AK_debmod_state* AK_debmod_init (
            void  )
```

Initializes debug mode structure [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

initialized debug mode state

5.11.2.18 AK_debmod_leave_critical_sec()

```
void AK_debmod_leave_critical_sec (
            AK_debmod_state * ds )
```

Makes ds available [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ds	debug mode state

Returns

void

5.11.2.19 AK_debmod_log_memory_alloc()

```
void AK_debmod_log_memory_alloc (
            int32_t func_id )
```

print debmod information on function [private function]

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func↩ _id	calling function id

Returns

void

5.11.2.20 AK_debmod_print_function_use()

```
void AK_debmod_print_function_use (
            const char * func_name,
            uint8_t in_recur )
```

Print function dependency [private function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name
in_recur	called in recursion

Returns

void

5.11.2.21 AK_free()

```
void AK_free (
            void * ptr )
```

Free memory at ptr (see free) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ptr	pointer to memory

Returns

void

5.11.2.22 AK_malloc()

```
void* AK_malloc (
            size_t size )
```

Allocate memory (see malloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

size	of memory to allocate in bytes

Returns

allocated memory or NULL

5.11.2.23 AK_mempro_test()

```
void AK_mempro_test ( )
```

Test function.

Author

Ivan Kristo

5.11.2.24 AK_print_active_functions()

```
void AK_print_active_functions ( )
```

Print all detected functions.

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.11.2.25 AK_print_function_use()

```
void AK_print_function_use (
            const char * func_name )
```

Print function dependency [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

func_name	function name

Returns

void

5.11.2.26 AK_print_function_uses()

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

Author

Marin Rukavina, Mislav Bozicevic

Returns

void

5.11.2.27 AK_realloc()

```
void* AK_realloc (
            void * ptr,
            size_t size )
```

Reallocates memory (see realloc) [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

ptr	old memory
size	new size

Returns

reallocated memory or NULL

5.11.2.28 AK_write_protect()

```
void AK_write_protect (
            void * memory )
```

Function write-protects memory [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

memory	

Returns

void

5.11.2.29 AK_write_unprotect()

```
void AK_write_unprotect (
            void * memory )
```

Function write-unprotects memory [public function].

Author

Marin Rukavina, Mislav Bozicevic

Parameters

memory	

Returns

void

5.12 auxi/observable.c File Reference

```
#include "./observable.h"
```

Include dependency graph for observable.c:

Classes

- struct _notifyDetails
- struct TypeObservable
- struct TypeObserver

Typedefs

- typedef struct _notifyDetails **NotifyDetails**
- typedef struct TypeObservable **AK_TypeObservable**
- typedef struct TypeObserver **AK_TypeObserver**
- typedef struct TypeObserver **AK_TypeObserver_Second**

Enumerations

- enum **NotifyType** { **ERROR**, **INFO**, **WARMING** }

Functions

- AK_observable * AK_init_observable (void *AK_observable_type, AK_ObservableType_Enum AK_↩ ObservableType_Def, void *AK_custom_action)

  Function that initializes a observable object.
- AK_observer * AK_init_observer (void *observer_type, void(*observer_type_event_handler)(void *, void *, AK_ObservableType_Enum))

  Function that initializes the observer object.
- char * **AK_get_message** (AK_TypeObservable *self)
- int **AK_custom_register_observer** (AK_TypeObservable *self, AK_observer *observer)
- int **AK_custom_unregister_observer** (AK_TypeObservable *self, AK_observer *observer)
- void **AK_set_notify_info_details** (AK_TypeObservable *self, NotifyType type, char *message)
- int **AK_custom_action** (void *data)
- AK_TypeObservable * **init_observable_type** ()
- void **handle_AK_custom_type** (AK_TypeObserver *observer, AK_TypeObservable *observable)
- void **custom_observer_event_handler** (void *observer, void *observable, AK_ObservableType_Enum A↩ K_ObservableType_Def)
- AK_TypeObserver * **init_observer_type** (void *observable)
- AK_TypeObserver * **init_observer_type_second** ()
- TestResult AK_observable_test ()

  Function that runs tests for observable pattern.
- TestResult **AK_observable_pattern** ()

5.12.1 Detailed Description

File that provides the implementations of functions for observable pattern

5.12.2 Function Documentation

5.12.2.1 AK_init_observable()

```
AK_observable* AK_init_observable (
            void * AK_observable_type,
            AK_ObservableType_Enum AK_ObservableType_Def,
            void * AK_custom_action )
```

Function that initializes a observable object.

Author

Ivan Pusic

Returns

Pointer to new observable object

5.12.2.2 AK_init_observer()

```
AK_observer* AK_init_observer (
            void * observer_type,
            void(*)(void *, void *, AK_ObservableType_Enum) observer_type_event_handler )
```

Function that initializes the observer object.

Author

Ivan Pusic

Returns

Pointer to new observer object

5.12.2.3 AK_observable_test()

```
TestResult AK_observable_test ( )
```

Function that runs tests for observable pattern.

Author

Ivan Pusic

5.13 auxi/observable.h File Reference

```
#include "test.h"
#include "constants.h"
#include "debug.h"
#include "mempro.h"
#include <string.h>
```

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

Classes

- struct Observer

  Structure that defines the functions for observer object.
- struct Observable

  Structure that defines the functions for observable object.

Typedefs

- typedef struct Observer **AK_observer**
- typedef struct Observable **AK_observable**

Enumerations

- enum **AK_ObservableType_Enum** { **AK_TRANSACTION**, **AK_TRIGGER**, **AK_CUSTOM_FIRST**, **AK_C↩ USTOM_SECOND** }

Functions

- AK_observer ∗ AK_init_observer (void ∗observable_type, void(∗observable_type_event_handler)(void ∗, void ∗, AK_ObservableType_Enum))

  Function that initializes the observer object.
- AK_observable ∗ AK_init_observable (void ∗AK_observable_type, AK_ObservableType_Enum AK_↩ ObservableType_Def, void ∗AK_custom_action)

  Function that initializes a observable object.
- TestResult AK_observable_test ()

  Function that runs tests for observable pattern.
- TestResult **AK_observable_pattern** ()

5.13.1 Detailed Description

Header file that provides data structures and declarations of functions for observable pattern

5.13.2 Function Documentation

5.13.2.1 AK_init_observable()

```
AK_observable* AK_init_observable (
            void * AK_observable_type,
            AK_ObservableType_Enum AK_ObservableType_Def,
            void * AK_custom_action )
```

Function that initializes a observable object.

Author

Ivan Pusic

Returns

Pointer to new observable object

5.13.2.2 AK_init_observer()

```
AK_observer* AK_init_observer (
            void * observer_type,
            void(*)(void *, void *, AK_ObservableType_Enum) observer_type_event_handler )
```

Function that initializes the observer object.

Author

Ivan Pusic

Returns

Pointer to new observer object

5.13.2.3 AK_observable_test()

```
TestResult AK_observable_test ( )
```

Function that runs tests for observable pattern.

Author

Ivan Pusic

5.14 auxi/test.c File Reference

```
#include "test.h"
```

Include dependency graph for test.c:

Functions

- TestResult TEST_result (int successfulAmount, int failedAmount)

  Returns the amount of successful and failed tests.
- void TEST_output_results (TestResult result)

  Prints a beautiful string informing the user of test results in the terminal.

5.14.1 Detailed Description

Provides functions for reporting test results for modules.

5.14.2 Function Documentation

5.14.2.1 TEST_output_results()

```
void TEST_output_results (
            TestResult result )
```

Prints a beautiful string informing the user of test results in the terminal.

Author

Igor Rinkovec

Returns

void

5.14.2.2 TEST_result()

```
TestResult TEST_result (
            int successfulAmount,
            int failedAmount )
```

Returns the amount of successful and failed tests.

Author

Igor Rinkovec

Parameters

successfulAmount	amount of successful tests
failedAmount	amount of failed tests

Returns

TestResult

5.15 file/test.c File Reference

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

Include dependency graph for test.c:

Functions

- char * AK_get_table_atribute_types (char *tblName)
  returns a string containing attribute types for the 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 the 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)
  Function that prints the requested column.
- int get_row_test (int num, char *tbl)
  Function that prints the requested row.
- void AK_create_test_tables ()
  Function for creating test tables.

5.15.1 Detailed Description

Provides functions for testing purposes

5.15.2 Function Documentation

5.15.2.1 AK_create_test_tables()

```
void AK_create_test_tables ( )
```

Function for creating test tables.

Author

Dino Laktašić

Returns

No return value

5.15.2.2 AK_get_table_atribute_types()

```
char* AK_get_table_atribute_types (
            char * tblName )
```

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

Author

Goran Štrok

Parameters

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

5.15.2.3 create_header_test()

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

Function for creating test table header.

Author

Luka Rajcevic

Parameters

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

Returns

1 if ok, 0 otherwise

5.15.2.4 get_column_test()

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

Function that prints the requested column.

Author

Luka Rajcevic

Returns

1 if column is found, 0 otherwise

Parameters

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

5.15.2.5 get_row_test()

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

Function that prints the requested row.

Author

Luka Rajcevic

Returns

1 if row is found, 0 otherwise

Parameters

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

5.15.2.6 insert_data_test()

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

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

Author

Luka Rajcevic

Parameters

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

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.15.2.7 selection_test()

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

Function for selection operator on one table.

Author

Luka Rajcevic

-

Parameters

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

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.16 file/test.h File Reference

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

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

Functions

- char * AK_get_table_atribute_types (char *tblName)
  returns a string containing attribute types for the 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 the 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)
  Function that prints the requested column.
- int get_row_test (int num, char *tbl)
  Function that prints the requested row.
- void AK_create_test_tables ()
  Function for creating test tables.

5.16.1 Detailed Description

Header file that provides functions and defines for testing purposes

5.16.2 Function Documentation

5.16.2.1 AK_create_test_tables()

```
void AK_create_test_tables ( )
```

Function for creating test tables.

Author

Dino Laktašić

Returns

No return value

5.16.2.2 AK_get_table_atribute_types()

```
char* AK_get_table_atribute_types (
            char * tblName )
```

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

Author

Goran Štrok

Parameters

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

5.16.2.3 create_header_test()

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

Function for creating test table header.

Author

Luka Rajcevic

Parameters

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

Returns

1 if ok, 0 otherwise

5.16.2.4 get_column_test()

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

Function that prints the requested column.

Author

Luka Rajcevic

Returns

1 if column is found, 0 otherwise

Parameters

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

5.16.2.5 get_row_test()

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

Function that prints the requested row.

Author

Luka Rajcevic

Returns

1 if row is found, 0 otherwise

Parameters

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

5.16.2.6 insert_data_test()

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

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

Author

Luka Rajcevic

Parameters

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

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.16.2.7 selection_test()

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

Function for selection operator on one table.

Author

Luka Rajcevic

-

Parameters

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

Returns

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

5.17 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 *allocationSet, int fromWhere, int gaplength, int numRequestedBlocks, AK_allocation_set_mode mode, int target)

  Function prepare demanded sets from allocation table.
- int AK_allocationtable_dump (int verbosity)

  Dumps the allocation table from the global allocation bit-vector onto standard output.
- void AK_blocktable_dump (int verbosity)

  Dumps the bit-table from the global allocation bit-vector onto standard output.
- int AK_blocktable_flush ()

  Function flushes bitmask table to the disk.
- void AK_allocate_block_activity_modes ()

  Allocation of an 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.
- int AK_blocktable_get ()

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

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

  Function that initializes the allocation table, writes it to the disk and caches it 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 a 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 updates the 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 that writes a block to the 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 *blockSet, int blockSetSize)

  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 that 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 that allocates a new blocks for increasing extent size.
- int AK_new_extent (int start_address, int old_size, int extent_type, AK_header *header)

  Function that 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_E↩ XTENT_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 *attribute_name, int type, int integrity, char *constr_name, char *contr_code)

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

  Function that inserts an 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 that 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 that 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 that 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 that deletes an extent between the first and the last block.
- int AK_delete_segment (char *name, int type)
- int AK_init_disk_manager ()
- TestResult **AK_allocationbit_test** ()
- TestResult **AK_allocationtable_test** ()
- TestResult 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.

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 containted in this wariables. If they don't match, then the error occured! 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 suceeded?" 0 means NO, 1 means YES.

5.17.1 Detailed Description

Defines functions for the disk manager

5.17.2 Function Documentation

5.17.2.1 AK_allocate_block_activity_modes()

```
void AK_allocate_block_activity_modes ( )
```

Allocation of an 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

5.17.2.2 AK_allocate_blocks()

```
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 updates the last initialized index.

Author

Markus Schatten , rearranged by dv

Returns

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

5.17.2.3 AK_allocationtable_dump()

```
int AK_allocationtable_dump (
            int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

Author

dv

Parameters

verbosity	level of verbosity (1 - minimal, 0 - no output)

5.17.2.4 AK_blocktable_dump()

```
void AK_blocktable_dump (
            int verbosity )
```

Dumps the bit-table from the global allocation bit-vector onto standard output.

Author

dv

Parameters

verbosity	level of verbosity (1 - verbose, 0 - minimal)

5.17.2.5 AK_blocktable_flush()

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to the disk.

Author

dv

Returns

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

5.17.2.6 AK_blocktable_get()

```
int AK_blocktable_get ( )
```

Function gets allocation table from the disk.

Author

dv

Returns

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

5.17.2.7 AK_copy_header()

```
int AK_copy_header (
            AK_header * header,
            int * blockSet,
            int blockSetSize )
```

Function copy header to blocks. Completely thread-safe.

Author

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

Parameters

header	Pointer to header which will be copied into each block in blockSet
blockSet	Pointer to array of block addresses into which to copy header
blockSetSize	Number of blocks in blockSet

Returns

number of performed header copy

5.17.2.8 AK_create_header()

```
AK_header* AK_create_header (
            char * attribute_name,
            int type,
            int integrity,
            char * constr_name,
            char * contr_code )
```

Function that creates header and initalize integrity, constraint name and constraint code with parameter values of function.

Author

Matija Novak

Parameters

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

Returns

AK_header

5.17.2.9 AK_delete_block()

```
int AK_delete_block (
            int address )
```

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

Author

Markus Schatten

Parameters

address	address of the block to be deleted

Returns

returns EXIT_SUCCESS if deletion successful, else EXIT_ERROR

5.17.2.10 AK_delete_extent()

```
int AK_delete_extent (
            int begin,
            int end )
```

Function that deletes an extent between the first and the last block.

Author

Dejan Sambolić

Parameters

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

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.17.2.11 AK_delete_segment()

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

Author

Mislav Èakariæ

Parameters

name	name of the segment
type	type of the segment

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.17.2.12 AK_get_allocation_set()

```
int AK_get_allocation_set (
            int * allocationSet,
            int fromWhere,
            int gaplength,
            int numRequestedBlocks,
            AK_allocation_set_mode mode,
            int target )
```

Function prepare demanded sets from allocation table.

Author

dv

Parameters

allocationSet	Pointer to array which will be filled and represent the allocation set
fromWhere	Has meaning only if mode is SEQUENCE. It describes from which address searching starts.
gaplength	Tells how many used blocks can be tolerated in allocation set
numRequestedBlocks	Tells how many AK_free blocks have been requested
mode	Defines how to obtain set of indexes to AK_free addresses
target	Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides

Returns

the first element of the allocation set

5.17.2.13 AK_get_extent()

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

Author

dv

Parameters

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

Returns

pointer to set of alocated block addresses

vars for loop [for]

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

5.17.2.14 AK_increase_extent()

```
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 that allocates a new blocks for increasing extent size.

Author

dv

Parameters

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

Returns

pointer to set of alocated block addresses

5.17.2.15 AK_init_allocation_table()

```
int AK_init_allocation_table ( )
```

Function that initializes the allocation table, writes it to the disk and caches it in memory.

Author

dv

Returns

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

5.17.2.16 AK_init_block()

```
AK_block* AK_init_block ( )
```

Function that initializes new block.

Author

Markus Schatten , rearranged by dv

Returns

pointer to block allocated in memory

5.17.2.17 AK_init_db_file()

```
int AK_init_db_file (
            int size )
```

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

Author

Markus Schatten

Parameters

size	size of new file in in blocks

Returns

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

5.17.2.18 AK_init_disk_manager()

```
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.

5.17.2.19 AK_init_system_catalog()

```
int AK_init_system_catalog ( )
```

Function that 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

5.17.2.20 AK_init_system_tables_catalog()

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

Author

Matija Novak

Parameters

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

Returns

EXIT_SUCCESS if initialization was succesful if not returns EXIT_ERROR

first header attribute of catalog_block

second attribute of catalog_block

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

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

merge catalog_heder with heders created before

5.17.2.21 AK_insert_entry()

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

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

Author

Matija Novak

Parameters

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

Returns

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

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

5.17.2.22 AK_memset_int()

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

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

Author

Miroslav Policki

Parameters

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

Returns

No return value

5.17.2.23 AK_new_extent()

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

Function that 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 Laktašiæ (fixed header BUG), refurbished by dv

Parameters

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

Returns

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

5.17.2.24 AK_new_segment()

```
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 INI↩ TIAL_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_EXTE↩ NT_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK_free.

Author

Tomislav Fotak, refurbished by dv

Parameters

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

Returns

EXIT_SUCCESS for success or EXIT_ERROR if some error occurs

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

5.17.2.25 AK_print_block()

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

Function that dumps a block.

Author

dv

Returns

nothing

5.17.2.26 AK_read_block()

```
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 by dv and Domagoj Šitum (thread-safe enabled)

Parameters

address	block number (address)

Returns

pointer to block allocated in memory

5.17.2.27 AK_read_block_for_testing()

```
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

5.17.2.28 AK_register_system_tables()

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

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

Author

Unknown

Parameters

relation	relation in database
attribute	attribute in databse
index	index in database
view	view in database
sequence	sequence in database
function	function in database

Parameters

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

Returns

EXIT_SUCCESS

5.17.2.29 AK_thread_safe_block_access_test()

```
TestResult 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

5.17.2.30 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

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

Function that writes the new value in block when index is updated.

Author

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

Parameters

block	poiner to block allocated in memory to write

Returns

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

5.17.2.31 AK_write_block_for_testing()

```
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

5.17.2.32 fsize()

```
int fsize (
            FILE * fp )
```

Helper function to determine file size.

Returns

file size

5.18 dm/dbman.h File Reference

```
#include "../auxi/test.h"
#include "../auxi/auxiliary.h"
#include <errno.h>
#include <pthread.h>
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../auxi/mempro.h"
#include <limits.h>
```

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

Classes

- struct AK_header

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

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

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

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

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

Macros

- #define **BITMASK**(b) (1 << ((b) % CHAR_BIT))
- #define **BITSLOT**(b) ((int)((b) / CHAR_BIT))
- #define **BITSET**(a, b) ((a)[BITSLOT(b)] |= BITMASK(b))
- #define **BITCLEAR**(a, b) ((a)[BITSLOT(b)] &= ~BITMASK(b))
- #define **BITTEST**(a, b) ((a)[BITSLOT(b)] & BITMASK(b))
- #define **BITNSLOTS**(nb) ((int)(nb + CHAR_BIT - 1) / CHAR_BIT)
- #define **SEGMENTLENGTH**() (BITNSLOTS(DB_FILE_BLOCKS_NUM) + 2*sizeof(int))
- #define **DB_FILE_SIZE_EX** 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 od allocation table LOWER - set tries to place itself to lower part od allocation table AROUND - set tries to place itself around targeted index.

Functions

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

  Function that dumps a block.
- TestResult **AK_allocationbit_test** ()
- TestResult **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 that allocates a 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 that 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 updates the last initialized index.
- AK_block * AK_init_block ()

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

  Dumps the allocation table from the global allocation bit-vector onto standard output.
- void AK_blocktable_dump (int zz)

  Dumps the bit-table from the global allocation bit-vector onto standard output.
- int AK_blocktable_flush ()

  Function flushes bitmask table to the disk.
- TestResult 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 the disk.
- int fsize (FILE *fp)

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

  Function that initializes the allocation table, writes it to the disk and caches it 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 that writes a block to the 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 that 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_E↩ XTENT_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL_EXTENT_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK_free.

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

  Function that creates header and initalize integrity, constraint name and constraint code with parameter values of function.

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

  Function that inserts an 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 that 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 that 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 that 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 that deletes an extent between the first and the last block.

- 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**

5.18.1 Detailed Description

Header file that contains all defines, includes and data structures for the disk manager of Kalashnikov DB

5.18.2 Macro Definition Documentation

5.18.2.1 AK_ALLOCATION_TABLE_SIZE

```
#define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)
```

Holds size of allocation table.

Author

dv

5.18.2.2 CHAR_IN_LINE

```
#define CHAR_IN_LINE 80
```

How many characters could line contain.

Author

dv

5.18.2.3 MAX_BLOCK_INIT_NUM

```
#define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY
```

How many blocks would be initially allocated.

Author

dv

5.18.3 Enumeration Type Documentation

5.18.3.1 AK_allocation_set_mode

```
enum AK_allocation_set_mode
```

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

Author

dv

5.18.4 Function Documentation

5.18.4.1 AK_allocate_blocks()

```
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 updates the last initialized index.

Author

Markus Schatten , rearranged by dv

Returns

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

5.18.4.2 AK_allocationtable_dump()

```
int AK_allocationtable_dump (
            int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

Author

dv

Parameters

verbosity	level of verbosity (1 - minimal, 0 - no output)

5.18.4.3 AK_blocktable_dump()

```
void AK_blocktable_dump (
            int verbosity )
```

Dumps the bit-table from the global allocation bit-vector onto standard output.

Author

dv

Parameters

verbosity	level of verbosity (1 - verbose, 0 - minimal)

5.18.4.4 AK_blocktable_flush()

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to the disk.

Author

dv

Returns

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

5.18.4.5 AK_blocktable_get()

```
int AK_blocktable_get ( )
```

Function gets allocation table from the disk.

Author

dv

Returns

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

5.18.4.6 AK_copy_header()

```
int AK_copy_header (
            AK_header * header,
            int * blockSet,
            int blockSetSize )
```

Function copy header to blocks. Completely thread-safe.

Author

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

Parameters

header	Pointer to header which will be copied into each block in blockSet
blockSet	Pointer to array of block addresses into which to copy header
blockSetSize	Number of blocks in blockSet

Returns

number of performed header copy

5.18.4.7 AK_create_header()

```
AK_header* AK_create_header (
            char * attribute_name,
            int type,
            int integrity,
            char * constr_name,
            char * contr_code )
```

Function that creates header and initalize integrity, constraint name and constraint code with parameter values of function.

Author

Matija Novak

Parameters

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

Returns

AK_header

5.18.4.8 AK_delete_block()

```
int AK_delete_block (
            int address )
```

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

Author

Markus Schatten

Parameters

address	address of the block to be deleted

Returns

returns EXIT_SUCCESS if deletion successful, else EXIT_ERROR

5.18.4.9 AK_delete_extent()

```
int AK_delete_extent (
            int begin,
            int end )
```

Function that deletes an extent between the first and the last block.

Author

Dejan Sambolić

Parameters

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

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.18.4.10 AK_delete_segment()

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

Author

Mislav Èakariæ

Parameters

name	name of the segment
type	type of the segment

Returns

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

5.18.4.11 AK_get_allocation_set()

```
int AK_get_allocation_set (
            int * allocationSet,
            int fromWhere,
            int gaplength,
            int numRequestedBlocks,
            AK_allocation_set_mode mode,
            int target )
```

Function prepare demanded sets from allocation table.

Author

dv

Parameters

allocationSet	Pointer to array which will be filled and represent the allocation set
fromWhere	Has meaning only if mode is SEQUENCE. It describes from which address searching starts.
gaplength	Tells how many used blocks can be tolerated in allocation set
numRequestedBlocks	Tells how many AK_free blocks have been requested
mode	Defines how to obtain set of indexes to AK_free addresses
target	Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides

Returns

the first element of the allocation set

5.18.4.12 AK_get_extent()

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

Author

dv

Parameters

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

Returns

pointer to set of alocated block addresses

vars for loop [for]

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

5.18.4.13 AK_increase_extent()

```
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 that allocates a new blocks for increasing extent size.

Author

dv

Parameters

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

Returns

pointer to set of alocated block addresses

5.18.4.14 AK_init_allocation_table()

```
int AK_init_allocation_table ( )
```

Function that initializes the allocation table, writes it to the disk and caches it in memory.

Author

dv

Returns

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

5.18.4.15 AK_init_block()

```
AK_block* AK_init_block ( )
```

Function that initializes new block.

Author

Markus Schatten , rearranged by dv

Returns

pointer to block allocated in memory

5.18.4.16 AK_init_db_file()

```
int AK_init_db_file (
            int size )
```

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

Author

Markus Schatten

Parameters

size	size of new file in in blocks

Returns

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

5.18.4.17 AK_init_disk_manager()

```
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.

5.18.4.18 AK_init_system_catalog()

```
int AK_init_system_catalog ( )
```

Function that 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

5.18.4.19 AK_init_system_tables_catalog()

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

Author

Matija Novak

Parameters

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

Returns

EXIT_SUCCESS if initialization was succesful if not returns EXIT_ERROR

first header attribute of catalog_block

second attribute of catalog_block

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

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

merge catalog_heder with heders created before

5.18.4.20 AK_insert_entry()

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

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

Author

Matija Novak

Parameters

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

Returns

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

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

5.18.4.21 AK_memset_int()

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

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

Author

Miroslav Policki

Parameters

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

Returns

No return value

5.18.4.22 AK_new_extent()

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

Function that 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 Laktašiæ (fixed header BUG), refurbished by dv

Parameters

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

Returns

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

5.18.4.23 AK_new_segment()

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

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

Author

Tomislav Fotak, refurbished by dv

Parameters

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

Returns

EXIT_SUCCESS for success or EXIT_ERROR if some error occurs

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

5.18.4.24 AK_print_block()

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

Function that dumps a block.

Author

dv

Returns

nothing

5.18.4.25 AK_read_block()

```
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 by dv and Domagoj Šitum (thread-safe enabled)

Parameters

address	block number (address)

Returns

pointer to block allocated in memory

5.18.4.26 AK_read_block_for_testing()

```
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

5.18.4.27 AK_register_system_tables()

```
int AK_register_system_tables (
            int relation,
            int attribute,
            int index,
            int view,
            int sequence,
            int function,
            int function_arguments,
            int trigger,
            int trigger_conditions,
            int db,
```

```
int db_obj,
int user,
int group,
int user_group,
int user_right,
int group_right,
int constraint,
int constraintNull,
int constraintCheck,
int constraintUnique,
int reference )
```

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

Author

Unknown

Parameters

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

Returns

EXIT_SUCCESS

5.18.4.28 AK_thread_safe_block_access_test()

```
TestResult 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

5.18.4.29 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

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

Author

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

Parameters

block	poiner to block allocated in memory to write

Returns

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

5.18.4.30 AK_write_block_for_testing()

```
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

5.18.4.31 fsize()

```
int fsize (
            FILE * fp )
```

Helper function to determine file size.

Returns

file size

5.18.5 Variable Documentation

5.18.5.1 AK_allocationbit

`AK_allocationbit`

Global variable that holds allocation bit-vector.

Author

dv

5.18.5.2 db

`db`

Variable that defines the DB file file handle.

Author

Markus Schatten

5.18.5.3 db_file_size

`db_file_size`

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

Author

Markus Schatten

5.19 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 that generates GUID.
- int AK_folder_exists (char ∗foldername)

  Function that checks if folder blobs already exists.
- int AK_mkdir (const char ∗path)

  Function that creates 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 that checks if folder blobs exists.
- void AK_split_path_file (char ∗∗p, char ∗∗f, char ∗pf)

  Function that splits a 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 that imports large objects to database.
- int AK_lo_export (char ∗oid, char ∗filepath)

  Function that retrieves large objects.
- int AK_lo_unlink (char ∗oid)

  Function that deletes large objects.
- TestResult AK_lo_test ()

  Tests.

Variables

- int **success** = 0
- int **failed** = 0

5.19.1 Detailed Description

Provides functions for manipulations of binary large objects

5.19.2 Function Documentation

5.19.2.1 AK_check_folder_blobs()

```
int AK_check_folder_blobs ( )
```

Function that checks if folder blobs exists.

Author

Samuel Picek

Returns

OID (object ID)

5.19.2.2 AK_concat()

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

Function for AK_concatinating 2 strings.

Author

Samuel Picek

Returns

returns new string

5.19.2.3 AK_folder_exists()

```
int AK_folder_exists (
            char * foldername )
```

Function that checks if folder blobs already exists.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.19.2.4 AK_GUID()

```
char* AK_GUID ( )
```

Function that generates GUID.

Author

Samuel Picek

Returns

returns globaly universal identifier based on kernel implementation

5.19.2.5 AK_lo_export()

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

Function that retrieves large objects.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.19.2.6 AK_lo_import()

```
char* AK_lo_import (
             char * filepath )
```

Function that imports large objects to database.

Author

Samuel Picek

Returns

OID (object ID)

5.19.2.7 AK_lo_test()

```
TestResult AK_lo_test ( )
```

Tests.

Author

Samuel Picek

5.19.2.8 AK_lo_unlink()

```
int AK_lo_unlink (
            char * oid )
```

Function that deletes large objects.

Author

Samuel Picek

Returns

OID (object ID)

5.19.2.9 AK_mkdir()

```
int AK_mkdir (
            const char * path )
```

Function that creates new folder.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.19.2.10 AK_split_path_file()

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

Function that splits a path from filename.

Author

Samuel Picek

Returns

void

5.20 file/blobs.h File Reference

```
#include "../auxi/test.h"
#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 that creates 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 that splits a path from filename.
- char * AK_GUID ()

  Function that generates GUID.
- int AK_folder_exists (char *foldername)

  Function that checks if folder blobs already exists.
- int AK_check_folder_blobs ()

  Function that checks 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 that imports large objects to database.
- int AK_lo_export (char *oid, char *filepath)

  Function that retrieves large objects.
- int AK_lo_unlink (char *oid)

  Function that deletes large objects.
- TestResult AK_lo_test ()

  Tests.

5.20.1 Detailed Description

Provides data structures, functions and defines for manipulating blobs

5.20.2 Function Documentation

5.20.2.1 AK_check_folder_blobs()

```
int AK_check_folder_blobs ( )
```

Function that checks if folder blobs exists.

Author

Samuel Picek

Returns

OID (object ID)

5.20.2.2 AK_concat()

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

Function for AK_concatinating 2 strings.

Author

Samuel Picek

Returns

returns new string

5.20.2.3 AK_folder_exists()

```
int AK_folder_exists (
            char * foldername )
```

Function that checks if folder blobs already exists.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.20.2.4 AK_GUID()

```
char* AK_GUID ( )
```

Function that generates GUID.

Author

Samuel Picek

Returns

returns globaly universal identifier based on kernel implementation

5.20.2.5 AK_lo_export()

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

Function that retrieves large objects.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.20.2.6 AK_lo_import()

```
char* AK_lo_import (
            char * filepath )
```

Function that imports large objects to database.

Author

Samuel Picek

Returns

OID (object ID)

5.20.2.7 AK_lo_test()

```
TestResult AK_lo_test ( )
```

Tests.

Author

Samuel Picek

5.20.2.8 AK_lo_unlink()

```
int AK_lo_unlink (
            char * oid )
```

Function that deletes large objects.

Author

Samuel Picek

Returns

OID (object ID)

5.20.2.9 AK_mkdir()

```
int AK_mkdir (
            const char * path )
```

Function that creates new folder.

Author

Samuel Picek

Returns

returns 0 for true and 1 for false

5.20.2.10 AK_split_path_file()

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

Function that splits a path from filename.

Author

Samuel Picek

Returns

void

5.21 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)

  !! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert_New_Element instead. 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_Update_Existing_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  Used to add a constraint attribute which will define what element gets updated when the operation is executed.
- void AK_Insert_New_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  Used to add a 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.
- TestResult **AK_fileio_test** ()

5.21.1 Detailed Description

Provides functions for file input/output

5.21.2 Function Documentation

5.21.2.1 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row @returs EXIT_SUCCESS if success

5.21.2.2 AK_delete_row_by_id()

```
void AK_delete_row_by_id (
            int id,
            char * tableName )
```

Function deletes row by id.

Author

Dražen Bandić

Parameters

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

5.21.2.3 AK_delete_row_from_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.

Author

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

Parameters

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

Returns

No return value

5.21.2.4 AK_delete_update_segment()

```
int AK_delete_update_segment (
            struct list_node * row_root,
            int del )
```

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

Author

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

Parameters

row_root	elements of one row
del	- DELETE or UPDATE

Returns

EXIT_SUCCESS if success

5.21.2.5 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
```

```
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

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

Returns

No return value

5.21.2.6 AK_Insert_New_Element_For_Update()

```
void AK_Insert_New_Element_For_Update (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore,
            int newconstraint )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

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

Returns

No return value

5.21.2.7 AK_insert_row()

```
int AK_insert_row (
            struct list_node * row_root )
```

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

Author

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

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.21.2.8 AK_insert_row_to_block()

```
int AK_insert_row_to_block (
            struct list_node * row_root,
            AK_block * temp_block )
```

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

Author

Matija Novak, updated by Dino Laktašić

Parameters

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

Returns

EXIT SUCCES if success

5.21.2.9 AK_Update_Existing_Element()

```
void AK_Update_Existing_Element (
             int newtype,
             void * data,
             char * table,
             char * attribute_name,
             struct list_node * ElementBefore )
```

Used to add a constraint attribute which will define what element gets updated when the operation is executed.

Author

Igor Rinkovec

Parameters

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

Returns

No return value

5.21.2.10 AK_update_row()

```
int AK_update_row (
             struct list_node * row_root )
```

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.21.2.11 AK_update_row_from_block()

```
void AK_update_row_from_block (
            AK_block * temp_block,
            struct list_node * row_root )
```

Function updates row from table in given block.

Author

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

Parameters

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

Returns

No return value

5.22 file/fileio.h File Reference

```
#include "../auxi/test.h"
#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 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)

  !! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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)

  Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.

- int AK_insert_row_to_block (struct list_node *row_root, AK_block *temp_block)

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

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

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

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

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

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

  Function updates rows of some table.
- TestResult **AK_fileio_test** ()
- void AK_delete_row_by_id (int id, char *tableName)

  Function deletes row by id.

5.22.1 Detailed Description

Header file provides functions and defines for file input/output

5.22.2 Function Documentation

5.22.2.1 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row @returs EXIT_SUCCESS if success

5.22.2.2 AK_delete_row_by_id()

```
void AK_delete_row_by_id (
            int id,
            char * tableName )
```

Function deletes row by id.

Author

Dražen Bandić

Parameters

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

5.22.2.3 AK_delete_row_from_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.

Author

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

Parameters

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

Returns

No return value

5.22.2.4 AK_delete_update_segment()

```
int AK_delete_update_segment (
            struct list_node * row_root,
            int del )
```

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

Author

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

Parameters

row_root	elements of one row
del	- DELETE or UPDATE

Returns

EXIT_SUCCESS if success

5.22.2.5 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

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

Returns

No return value

5.22.2.6 AK_Insert_New_Element_For_Update()

```
void AK_Insert_New_Element_For_Update (
            int newtype,
```

```
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore,
            int newconstraint )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

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

Returns

No return value

5.22.2.7 AK_insert_row()

```
int AK_insert_row (
            struct list_node * row_root )
```

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

Author

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

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.22.2.8 AK_insert_row_to_block()

```
int AK_insert_row_to_block (
            struct list_node * row_root,
            AK_block * temp_block )
```

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

Author

Matija Novak, updated by Dino Laktašić

Parameters

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

Returns

EXIT SUCCES if success

5.22.2.9 AK_update_row()

```
int AK_update_row (
            struct list_node * row_root )
```

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.22.2.10 AK_update_row_from_block()

```
void AK_update_row_from_block (
            AK_block * temp_block,
            struct list_node * row_root )
```

Function updates row from table in given block.

Author

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

Parameters

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

Returns

No return value

5.23 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 that initializes a 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 that initializes a new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- TestResult AK_files_test ()

  Test function.

Variables

- pthread_mutex_t **fileMut** = PTHREAD_MUTEX_INITIALIZER

5.23.1 Detailed Description

Header file provides functions for file management

5.23.2 Function Documentation

5.23.2.1 AK_files_test()

```
TestResult AK_files_test ( )
```

Test function.

Author

Unknown

Returns

No return value

5.23.2.2 AK_initialize_new_index_segment()

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

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

Author

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

Parameters

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

Returns

start address of new segment

5.23.2.3 AK_initialize_new_segment()

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

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

Author

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

Parameters

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

Returns

start address of new segment

5.24 file/files.h File Reference

```
#include "../auxi/test.h"
#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 that initializes a 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 that initializes a new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.
- TestResult AK_files_test ()

  Test function.

5.24.1 Detailed Description

Header file that provides functions and defines for file management

5.24.2 Function Documentation

5.24.2.1 AK_files_test()

```
TestResult AK_files_test ( )
```

Test function.

Author

Unknown

Returns

No return value

5.24.2.2 AK_initialize_new_index_segment()

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

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

Author

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

Parameters

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

Returns

start address of new segment

5.24.2.3 AK_initialize_new_segment()

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

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

Author

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

Parameters

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

Returns

start address of new segment

5.25 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)

  Function that 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 that deallocates memory used by the search result returned by AK_search_unsorted.
- TestResult AK_filesearch_test ()

  Function that tests file search.

5.25.1 Detailed Description

Provides functions for file searching

5.25.2 Function Documentation

5.25.2.1 AK_deallocate_search_result()

```
void AK_deallocate_search_result (
            search_result srResult )
```

Function that deallocates memory used by the search result returned by AK_search_unsorted.

Author

Miroslav Policki

Parameters

srResult	search result

Returns

No return value

5.25.2.2 AK_filesearch_test()

```
TestResult AK_filesearch_test ( )
```

Function that tests file search.

Author

Miroslav Policki

Returns

No return value

5.25.2.3 AK_search_unsorted()

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

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

Author

Miroslav Policki

Parameters

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

Returns

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

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

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

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

5.26 file/filesearch.h File Reference

```
#include "../auxi/test.h"
#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)

  Function that 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 that deallocates memory used by the search result returned by AK_search_unsorted.
- TestResult AK_filesearch_test ()

  Function that tests file search.

5.26.1 Detailed Description

Header file provides data structures, functions and defines for file searching

5.26.2 Function Documentation

5.26.2.1 AK_deallocate_search_result()

```
void AK_deallocate_search_result (
            search_result srResult )
```

Function that deallocates memory used by the search result returned by AK_search_unsorted.

Author

Miroslav Policki

Parameters

srResult	search result

Returns

No return value

5.26.2.2 AK_filesearch_test()

```
TestResult AK_filesearch_test ( )
```

Function that tests file search.

Author

Miroslav Policki

Returns

No return value

5.26.2.3 AK_search_unsorted()

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

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

Author

Miroslav Policki

Parameters

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

Returns

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

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

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

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

5.27 file/filesort.h File Reference

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

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

Macros

- #define DATA_ROW_SIZE 200

  Constatnt 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 that returns the total number of headers in the block.
- int AK_get_header_number (AK_block *iBlock, char *attribute_name)

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

  Function that returns tuples number in block.
- int AK_sort_segment (char *srcTable, char *destTable, struct list_node *attributes)

  Function that sorts a segment.
- void AK_reset_block (AK_block *block)

  Function that resets block.
- void AK_block_sort (AK_block *iBlock, char *atr_name)

  Function that sorts the given block.
- TestResult **AK_filesort_test** ()

5.27.1 Detailed Description

Header file that provides functions and defines for file sorting

5.27.2 Function Documentation

5.27.2.1 AK_block_sort()

```
void AK_block_sort (
            AK_block * iBlock,
            char * attribute_name )
```

Function that sorts the given block.

Author

Bakoš Nikola

Version

v1.0

Parameters

iBlock	block to be sorted

Returns

No return value

5.27.2.2 AK_get_header_number()

```
int AK_get_header_number (
            AK_block * iBlock,
            char * attribute_name )
```

Function that returns the 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]...

5.27.2.3 AK_get_num_of_tuples()

```
int AK_get_num_of_tuples (
            AK_block * iBlock )
```

Function that returns tuples number in block.

Author

Unknown

Returns

tuples number in block

5.27.2.4 AK_get_total_headers()

```
int AK_get_total_headers (
            AK_block * iBlock )
```

Function that returns the total number of headers in the block.

Author

Unknown

Returns

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

5.27.2.5 AK_reset_block()

```
void AK_reset_block (
            AK_block * block )
```

Function that resets block.

Author

Unknown

Parameters

block	block to be resetted

Returns

No return value

5.27.2.6 AK_sort_segment()

```
int AK_sort_segment (
            char * srcTable,
            char * destTable,
            struct list_node * attributes )
```

Function that sorts a segment.

Author

Tomislav Bobinac, updated by Filip Žmuk

Todo Make it to suport multiple sort atributes and ASC|DESC ordering

Returns

No return value.

Author

Tomislav Bobinac, updated by Filip Žmuk

Todo Make it to suport multiple sort atributes and ASC|DESC ordering

Returns

No return value.

5.28 file/id.c File Reference

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

Include dependency graph for id.c:

Functions

- int AK_get_id ()

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

  Function that fetches unique ID for any object, stored in sequence based on table name.
- TestResult AK_id_test ()

  Function for testing getting ID's.

5.28.1 Detailed Description

Provides functions for creating id of objects

5.28.2 Function Documentation

5.28.2.1 AK_get_id()

```
int AK_get_id ( )
```

Function that fetches unique ID for any object, stored in a sequence.

Author

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

Returns

objectID

5.28.2.2 AK_get_table_id()

```
char AK_get_table_id (
            char * tableName )
```

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

Author

Lovro Predovan

Returns

objectID in string(char) format

5.28.2.3 AK_id_test()

```
TestResult AK_id_test ( )
```

Function for testing getting ID's.

Author

Mislav Čakarić, updated by Nenad Makar

Returns

No retun value

5.29 file/id.h File Reference

```
#include "../auxi/test.h"
#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 that fetches unique ID for any object, stored in a sequence.
- TestResult AK_id_test ()

  Function for testing getting ID's.

5.29.1 Detailed Description

Provides functions and defines for creating id of objects

5.29.2 Function Documentation

5.29.2.1 AK_get_id()

```
int AK_get_id ( )
```

Function that fetches unique ID for any object, stored in a sequence.

Author

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

Returns

objectID

5.29.2.2 AK_id_test()

```
TestResult AK_id_test ( )
```

Function for testing getting ID's.

Author

Mislav Čakarić, updated by Nenad Makar

Returns

No retun value

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

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

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

  Function that gets adresses of the particuliar 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. That data is put in a list called add_root.
- void AK_print_Att_Test (list_ad *list)

  Function that prints the list of adresses.
- list_ad * AK_get_Attribute (char *tableName, char *attributeName, char *attributeValue)

  Function that fetches the values from the bitmap index if there is one for a 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 that updates the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.
- void AK_add_to_bitmap_index (char *tableName, char *attributeName)

  Function that writes the new value in block when index is updated.
- 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 the name of index.
- TestResult 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.

5.30.1 Detailed Description

Provides functions for bitmap indexes

5.30.2 Function Documentation

5.30.2.1 AK_add_to_bitmap_index()

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

Function that writes the new value in block when index is updated.

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

Author

Saša Vukšić

Parameters

block	block to write on

Returns

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

Author

Lovro Predovan

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected

Parameters

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

Returns

No return value

5.30.2.2 AK_bitmap_test()

```
TestResult 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

5.30.2.3 AK_create_Index()

```
void AK_create_Index (
            char * tblName,
            char * tblNameIndex,
            char * attributeName,
            int positionTbl,
            int numAtributes,
            AK_header * headerIndex )
```

Function that loads index table with the value of particulary atribute.

Author

Saša Vukšić, Lovro Predovan

Parameters

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

Returns

No return value

5.30.2.4 AK_create_Index_Table()

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

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

Author

Saša Vukšić, Lovro Predovan

Parameters

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

Returns

No return value

5.30.2.5 AK_delete_bitmap_index()

```
void AK_delete_bitmap_index (
            char * indexName )
```

Function that deletes bitmap index based on the name of index.

Author

Lovro Predovan

Parameters

Bitmap	index table name

Returns

No return value

5.30.2.6 AK_get_attribute()

```
list_ad* AK_get_attribute (
            char * indexName,
            char * attribute )
```

Function that gets adresses of the particuliar 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. That data is put in a list called add_root.

Author

Saša Vukšić, Lovro Predovan

Parameters

indexName	name of index
attribute	name of attribute

Returns

list of adresses

5.30.2.7 AK_get_Attribute()

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

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

Author

Saša Vukšić

Parameters

tableName	name of table
attributeValue	value of attribute

Returns

list of adresses

5.30.2.8 AK_If_ExistOp()

```
int AK_If_ExistOp (
             struct list_node * L,
             char * ele )
```

Function that examines whether list L contains operator ele.

Author

Saša Vukšić

Parameters

L	list of elements
ele	operator to be found in list

Returns

1 if operator ele is found in list, otherwise 0

5.30.2.9 AK_print_Att_Test()

```
void AK_print_Att_Test (
            list_ad * list )
```

Function that prints the list of adresses.

Author

Saša Vukšić, Lovro Predovan

Parameters

list	list of adresses

Returns

No return value

5.30.2.10 AK_print_Header_Test()

```
void AK_print_Header_Test (
            char * tblName )
```

Function that tests printing header of table.

Author

Saša Vukšić

Parameters

tblName	name of table who's header we are printing

Returns

No return value

5.30.2.11 AK_update()

```
void AK_update (
            int addBlock,
            int addTd,
            char * tableName,
```

```
char * attributeName,
char * attributeValue,
char * newAttributeValue )
```

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

Author

Saša Vukšić

Parameters

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

Returns

No return value

5.31 file/idx/bitmap.h File Reference

```
#include "../../auxi/test.h"
#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 that examines whether list L contains operator ele.
- void AK_create_Index_Table (char *tblName, struct list_node *attributes)

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

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

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

  Function that gets adresses of the particuliar 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. That data is put in a list called add_root.

- void **AK_create_List_Address_Test** ()
- void AK_print_Att_Test (list_ad ∗list)

  Function that prints the list of adresses.
- list_ad ∗ AK_get_Attribute (char ∗tableName, char ∗attributeName, char ∗attributeValue)

  Function that fetches the values from the bitmap index if there is one for a 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 that updates the index, only on values that alredy exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.
- int AK_write_block (AK_block ∗block)

  Function that writes the new value in block when index is updated.
- TestResult 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 the name of index.
- void AK_add_to_bitmap_index (char ∗tableName, char ∗attributeName)

  Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

5.31.1 Detailed Description

Header file that declares functions

5.31.2 Function Documentation

5.31.2.1 AK_add_to_bitmap_index()

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

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

Author

Lovro Predovan

Parameters

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

Returns

No return value

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

Author

Saša Vukšić

Parameters

block	block to write on

Returns

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

Author

Lovro Predovan

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected

Parameters

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

Returns

No return value

5.31.2.2 AK_bitmap_test()

```
TestResult 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

5.31.2.3 AK_create_Index()

```
void AK_create_Index (
            char * tblName,
            char * tblNameIndex,
            char * attributeName,
            int positionTbl,
            int numAtributes,
            AK_header * headerIndex )
```

Function that loads index table with the value of particulary atribute.

Author

Saša Vukšić, Lovro Predovan

Parameters

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

Returns

No return value

5.31.2.4 AK_create_Index_Table()

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

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

Author

Saša Vukšić, Lovro Predovan

Parameters

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

Returns

No return value

5.31.2.5 AK_delete_bitmap_index()

```
void AK_delete_bitmap_index (
            char * indexName )
```

Function that deletes bitmap index based on the name of index.

Author

Lovro Predovan

Parameters

Bitmap	index table name

Returns

No return value

5.31.2.6 AK_get_attribute()

```
list_ad* AK_get_attribute (
            char * indexName,
            char * attribute )
```

Function that gets adresses of the particuliar 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. That data is put in a list called add_root.

Author

Saša Vukšić, Lovro Predovan

Parameters

indexName	name of index
attribute	name of attribute

Returns

list of adresses

5.31.2.7 AK_get_Attribute()

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

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

Author

Saša Vukšić

Parameters

tableName	name of table
attributeValue	value of attribute

Returns

list of adresses

5.31.2.8 AK_If_ExistOp()

```
int AK_If_ExistOp (
             struct list_node * L,
             char * ele )
```

Function that examines whether list L contains operator ele.

Author

Saša Vukšić

Parameters

L	list of elements
ele	operator to be found in list

Returns

1 if operator ele is found in list, otherwise 0

5.31.2.9 AK_print_Att_Test()

```
void AK_print_Att_Test (
            list_ad * list )
```

Function that prints the list of adresses.

Author

Saša Vukšić, Lovro Predovan

Parameters

list	list of adresses

Returns

No return value

5.31.2.10 AK_print_Header_Test()

```
void AK_print_Header_Test (
            char * tblName )
```

Function that tests printing header of table.

Author

Saša Vukšić

Parameters

tblName	name of table who's header we are printing

Returns

No return value

5.31.2.11 AK_update()

```
void AK_update (
            int addBlock,
            int addTd,
            char * tableName,
```

```
            char * attributeName,
            char * attributeValue,
            char * newAttributeValue )
```

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

Author

Saša Vukšić

Parameters

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

Returns

No return value

5.31.2.12 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

Function that writes the new value in block when index is updated.

Author

Saša Vukšić

Parameters

block	block to write on

Returns

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

Function that writes the new value in block when index is updated.

Author

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

Parameters

block	poiner to block allocated in memory to write

Returns

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

5.32 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 that creates 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 that searches or deletes a value in btree index.
- int **AK_btree_insert** (char ∗indexName, int ∗insertValue, int ∗insertTd, int ∗insertBlock)
- TestResult **AK_btree_test** ()

5.32.1 Detailed Description

Header file that provides functions for BTree indices

5.32.2 Function Documentation

5.32.2.1 AK_btree_create()

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

Function that creates new btree index on integer attribute in table.

Author

Anđelko Spevec

Parameters

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

5.32.2.2 AK_btree_search_delete()

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

Function that searches or deletes a value in btree index.

Author

Anđelko Spevec

Parameters

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

5.33 file/idx/btree.h File Reference

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

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

Classes

- struct btree_node
- struct root_info

Macros

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

Functions

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

  Function that creates 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 that searches or deletes a value in btree index.
- int **AK_btree_insert** (char ∗indexName, int ∗insertValue, int ∗insertTd, int ∗insertBlock)
- TestResult **AK_btree_test** ()

5.33.1 Detailed Description

Header file that provides data strucures, functions and defines for BTree indices

5.33.2 Function Documentation

5.33.2.1 AK_btree_create()

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

Function that creates new btree index on integer attribute in table.

Author

Anđelko Spevec

Parameters

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

5.33.2.2 AK_btree_search_delete()

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

Function that searches or deletes a value in btree index.

Author

Anđelko Spevec

Parameters

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

5.34 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 that computes a hash value from varchar or integer.
- struct_add ∗ AK_insert_bucket_to_block (char ∗indexName, char ∗data, int type)

  Function that inserts a bucket to block.
- void AK_update_bucket_in_block (struct_add ∗add, char ∗data)

  Function that updates a bucket in block.
- void AK_change_hash_info (char ∗indexName, int modulo, int main_bucket_num, int hash_bucket_num)

  Function that changes a info of hash index.
- hash_info ∗ AK_get_hash_info (char ∗indexName)

  Function that fetches the info for hash index.
- struct_add ∗ AK_get_nth_main_bucket_add (char ∗indexName, int n)

  Function that fetches nth main bucket.
- void AK_insert_in_hash_index (char ∗indexName, int hashValue, struct_add ∗add)

  Function that inserts a record in hash bucket.
- struct_add ∗ AK_find_delete_in_hash_index (char ∗indexName, struct list_node ∗values, int delete)

  Function that fetches or deletes a record from hash index.
- struct_add ∗ AK_find_in_hash_index (char ∗indexName, struct list_node ∗values)

  Function that fetches a record from the hash index.
- void AK_delete_in_hash_index (char ∗indexName, struct list_node ∗values)

  Function that deletes a record from the hash index.
- int AK_create_hash_index (char ∗tblName, struct list_node ∗attributes, char ∗indexName)

  Function that creates a hash index.
- void **AK_delete_hash_index** (char ∗indexName)
- TestResult AK_hash_test ()

  Function that tests hash index.

5.34.1 Detailed Description

Provides functions for Hash indices

5.34.2 Function Documentation

5.34.2.1 AK_change_hash_info()

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

Function that changes a info of hash index.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.34.2.2 AK_create_hash_index()

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

Function that creates a hash index.

Author

Mislav Čakarić

Parameters

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

Returns

success or error

5.34.2.3 AK_delete_in_hash_index()

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

Function that deletes a record from the hash index.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.34.2.4 AK_elem_hash_value()

```
int AK_elem_hash_value (
            struct list_node * elem )
```

Function that computes a hash value from varchar or integer.

Author

Mislav Čakarić

Parameters

elem	element of row for wich value is to be computed

Returns

hash value

5.34.2.5 AK_find_delete_in_hash_index()

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

Function that fetches or deletes a record from hash index.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the record is in table

5.34.2.6 AK_find_in_hash_index()

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

Function that fetches a record from the hash index.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the record is in table

5.34.2.7 AK_get_hash_info()

```
hash_info* AK_get_hash_info (
            char * indexName )
```

Function that fetches the info for hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index

Returns

info bucket with info data for hash index

5.34.2.8 AK_get_nth_main_bucket_add()

```
struct_add* AK_get_nth_main_bucket_add (
            char * indexName,
            int n )
```

Function that fetches nth main bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
n	number of main bucket

Returns

address structure with data where the bucket is stored

5.34.2.9 AK_hash_test()

```
TestResult AK_hash_test ( )
```

Function that tests hash index.

Author

Mislav Čakarić

Returns

No return value

5.34.2.10 AK_insert_bucket_to_block()

```
struct_add* AK_insert_bucket_to_block (
            char * indexName,
            char * data,
            int type )
```

Function that inserts a bucket to block.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the bucket is stored

5.34.2.11 AK_insert_in_hash_index()

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

Function that inserts a record in hash bucket.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.34.2.12 AK_update_bucket_in_block()

```
void AK_update_bucket_in_block (
            struct_add * add,
            char * data )
```

Function that updates a bucket in block.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.35 file/idx/hash.h File Reference

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

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

Classes

- struct hash_info

Structure for defining a hash info element.

- struct bucket_elem

  Structure for defining a single bucket element.

- struct main_bucket

  Structure for defining main bucket for table hashing.

- struct hash_bucket

  Structure for hash bucket for table hashing.

Functions

- int AK_elem_hash_value (struct list_node *elem)

  Function that computes a hash value from varchar or integer.

- struct_add * AK_insert_bucket_to_block (char *indexName, char *data, int type)

  Function that inserts a bucket to block.

- void AK_update_bucket_in_block (struct_add *add, char *data)

  Function that updates a bucket in block.

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

  Function that changes a info of hash index.

- hash_info * AK_get_hash_info (char *indexName)

  Function that fetches the info for hash index.

- struct_add * AK_get_nth_main_bucket_add (char *indexName, int n)

  Function that fetches nth main bucket.

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

  Function that inserts a record in hash bucket.

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

  Function that fetches or deletes a record from hash index.

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

  Function that fetches a record from the hash index.

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

  Function that deletes a record from the hash index.

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

  Function that creates a hash index.

- void **AK_delete_hash_index** (char *indexName)
- TestResult AK_hash_test ()

  Function that tests hash index.

5.35.1 Detailed Description

Header file that provides data structures, functions and defines for Hash indices

5.35.2 Function Documentation

5.35.2.1 AK_change_hash_info()

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

Function that changes a info of hash index.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.35.2.2 AK_create_hash_index()

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

Function that creates a hash index.

Author

Mislav Čakarić

Parameters

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

Returns

success or error

5.35.2.3 AK_delete_in_hash_index()

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

Function that deletes a record from the hash index.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.35.2.4 AK_elem_hash_value()

```
int AK_elem_hash_value (
            struct list_node * elem )
```

Function that computes a hash value from varchar or integer.

Author

Mislav Čakarić

Parameters

elem	element of row for wich value is to be computed

Returns

hash value

5.35.2.5 AK_find_delete_in_hash_index()

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

Function that fetches or deletes a record from hash index.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the record is in table

5.35.2.6 AK_find_in_hash_index()

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

Function that fetches a record from the hash index.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the record is in table

5.35.2.7 AK_get_hash_info()

```
hash_info* AK_get_hash_info (
             char * indexName )
```

Function that fetches the info for hash index.

Author

Mislav Čakarić

Parameters

indexName	name of index

Returns

info bucket with info data for hash index

5.35.2.8 AK_get_nth_main_bucket_add()

```
struct_add* AK_get_nth_main_bucket_add (
            char * indexName,
            int n )
```

Function that fetches nth main bucket.

Author

Mislav Čakarić

Parameters

indexName	name of index
n	number of main bucket

Returns

address structure with data where the bucket is stored

5.35.2.9 AK_hash_test()

```
TestResult AK_hash_test ( )
```

Function that tests hash index.

Author

Mislav Čakarić

Returns

No return value

5.35.2.10 AK_insert_bucket_to_block()

```
struct_add* AK_insert_bucket_to_block (
            char * indexName,
            char * data,
            int type )
```

Function that inserts a bucket to block.

Author

Mislav Čakarić

Parameters

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

Returns

address structure with data where the bucket is stored

5.35.2.11 AK_insert_in_hash_index()

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

Function that inserts a record in hash bucket.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.35.2.12 AK_update_bucket_in_block()

```
void AK_update_bucket_in_block (
            struct_add * add,
            char * data )
```

Function that updates a bucket in block.

Author

Mislav Čakarić

Parameters

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

Returns

No return value

5.36 file/idx/index.c File Reference

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

Include dependency graph for index.c:

Functions

- void AK_InitializelistAd (list_ad *L)

  Function that initialises a linked list.
- element_ad AK_Get_First_elementAd (list_ad *L)

  Function that finds the first node of linked list.
- element_ad AK_Get_Last_elementAd (list_ad *L)

  Function that finds the last node of linked list.
- element_ad AK_Get_Next_elementAd (element_ad Currentelement_op)

  Function that finds the next node of a node in linked list.
- element_ad AK_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  Function that finds the previous node of a node in linked list.
- int AK_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad *L)

  Function that finds the position of a node in linked list.
- void AK_Delete_elementAd (element_ad Deletedelement_op, list_ad *L)

  Function that deletes a node from a linked list.
- void AK_Delete_All_elementsAd (list_ad *L)

  Function that deletes all nodes in a linked list.

- void AK_Insert_NewelementAd (int addBlock, int indexTd, char ∗attName, element_ad elementBefore)

  Function that inserts a new element into a linked list.
- int AK_num_index_attr (char ∗indexTblName)

  Function that fetches the number of elements in a 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 that 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 gets index table header.
- void AK_print_index_table (char ∗indexTblName)

  Function that prints out the index table.
- void AK_index_test ()

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

5.36.1 Detailed Description

Provides functions for indexes

5.36.2 Function Documentation

5.36.2.1 AK_Delete_All_elementsAd()

```
void AK_Delete_All_elementsAd (
            list_ad * L )
```

Function that deletes all nodes in a linked list.

Author

Unknown

Parameters

L	list head

Returns

No return value

5.36.2.2 AK_Delete_elementAd()

```
void AK_Delete_elementAd (
            element_ad Deletedelement_op,
            list_ad * L )
```

Function that deletes a node from a linked list.

Author

Unknown

Parameters

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

Returns

No return value

5.36.2.3 AK_Get_First_elementAd()

```
element_ad AK_Get_First_elementAd (
            list_ad * L )
```

Function that finds the first node of linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

Address of first node

5.36.2.4 AK_get_index_header()

```
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

∗*tblName*	table name

Returns

array of table header

5.36.2.5 AK_get_index_num_records()

```
int AK_get_index_num_records (
            char * indexTblName )
```

Determine number of rows in the table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

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

Parameters

∗*tableName*	table name

Returns

number of rows in the table

5.36.2.6 AK_get_index_tuple()

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

Function that gets value in some row and column.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

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

Returns

value in the list

5.36.2.7 AK_Get_Last_elementAd()

```
element_ad AK_Get_Last_elementAd (
            list_ad * L )
```

Function that finds the last node of linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

Address of last node or 0 if list is empty

5.36.2.8 AK_Get_Next_elementAd()

```
element_ad AK_Get_Next_elementAd (
            element_ad Currentelement_op )
```

Function that finds the next node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	address of current node

Returns

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

5.36.2.9 AK_Get_Position_Of_elementAd()

```
int AK_Get_Position_Of_elementAd (
            element_ad Searchedelement_op,
            list_ad * L )
```

Function that finds the position of a node in linked list.

Author

Unknown

Parameters

Searchedelement_op	address of current note
∗L	linked list head

Returns

Integer value of current node's order in the list

5.36.2.10 AK_Get_Previous_elementAd()

```
element_ad AK_Get_Previous_elementAd (
            element_ad Currentelement_op,
            element_ad L )
```

Function that finds the previous node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	Address of current node
L	previous element

Returns

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

5.36.2.11 AK_index_table_exist()

```
int AK_index_table_exist (
            char * indexTblName )
```

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

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

tblName	table name

Returns

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

5.36.2.12 AK_index_test()

```
void AK_index_test ( )
```

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

Author

Lovro Predovan

Returns

No return value

5.36.2.13 AK_InitializelistAd()

```
void AK_InitializelistAd (
            list_ad * L )
```

Function that initialises a linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

No return value

5.36.2.14 AK_Insert_NewelementAd()

```
void AK_Insert_NewelementAd (
            int addBlock,
            int indexTd,
            char * attName,
            element_ad elementBefore )
```

Function that inserts a new element into a linked list.

Author

Unknown

Parameters

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

Returns

No return value

5.36.2.15 AK_num_index_attr()

```
int AK_num_index_attr (
            char * indexTblName )
```

Function that fetches the number of elements in a index table.

Author

Lovro Predovan

Parameters

index	table name

Returns

No return value

5.36.2.16 AK_print_index_table()

```
void AK_print_index_table (
            char * indexTblName )
```

Function that prints out the index table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

∗*tblName*	table name

Returns

No return value

5.37 file/idx/index.h File Reference

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

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

Classes

- struct struct_add

  Structure defining node address.
- struct list_structure_ad

Typedefs

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

Functions

- int AK_index_table_exist (char ∗indexTblName)

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

  Function that prints out the 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 that fetches the number of elements in a index table.
- void AK_InitializelistAd (list_ad ∗L)

  Function that initialises a linked list.
- element_ad AK_Get_First_elementAd (list_ad ∗L)

  Function that finds the first node of linked list.
- element_ad AK_Get_Last_elementAd (list_ad ∗L)

  Function that finds the last node of linked list.
- element_ad AK_Get_Next_elementAd (element_ad Currentelement_op)

  Function that finds the next node of a node in linked list.
- element_ad AK_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  Function that finds the previous node of a node in linked list.
- int AK_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad ∗L)

  Function that finds the position of a node in linked list.
- void AK_Delete_elementAd (element_ad Deletedelement_op, list_ad ∗L)

  Function that deletes a node from a linked list.
- void AK_Delete_All_elementsAd (list_ad ∗L)

  Function that deletes all nodes in a linked list.
- void AK_Insert_NewelementAd (int addBlock, int indexTd, char ∗attName, element_ad elementBefore)

  Function that inserts a new element into a linked list.
- void AK_index_test ()

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

5.37.1 Detailed Description

Header file that provides data structures, functions and defines for bitmap index

5.37.2 Function Documentation

5.37.2.1 AK_Delete_All_elementsAd()

```
void AK_Delete_All_elementsAd (
            list_ad * L )
```

Function that deletes all nodes in a linked list.

Author

Unknown

Parameters

L	list head

Returns

No return value

5.37.2.2 AK_Delete_elementAd()

```
void AK_Delete_elementAd (
            element_ad Deletedelement_op,
            list_ad * L )
```

Function that deletes a node from a linked list.

Author

Unknown

Parameters

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

Returns

No return value

5.37.2.3 AK_Get_First_elementAd()

```
element_ad AK_Get_First_elementAd (
             list_ad * L )
```

Function that finds the first node of linked list.

Author

Unknown

Parameters

list_ad	*L linked list head

Returns

Address of first node

5.37.2.4 AK_get_index_num_records()

```
int AK_get_index_num_records (
             char * indexTblName )
```

Determine number of rows in the table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

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

Parameters

*tableName	table name

Returns

number of rows in the table

5.37.2.5 AK_get_index_tuple()

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

Function that gets value in some row and column.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

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

Returns

value in the list

5.37.2.6 AK_Get_Last_elementAd()

```
element_ad AK_Get_Last_elementAd (
            list_ad * L )
```

Function that finds the last node of linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

Address of last node or 0 if list is empty

5.37.2.7 AK_Get_Next_elementAd()

```
element_ad AK_Get_Next_elementAd (
            element_ad Currentelement_op )
```

Function that finds the next node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	address of current node

Returns

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

5.37.2.8 AK_Get_Position_Of_elementAd()

```
int AK_Get_Position_Of_elementAd (
            element_ad Searchedelement_op,
            list_ad * L )
```

Function that finds the position of a node in linked list.

Author

Unknown

Parameters

Searchedelement_op	address of current note
∗L	linked list head

Returns

Integer value of current node's order in the list

5.37.2.9 AK_Get_Previous_elementAd()

```
element_ad AK_Get_Previous_elementAd (
            element_ad Currentelement_op,
            element_ad L )
```

Function that finds the previous node of a node in linked list.

Author

Unknown

Parameters

Currentelement_op	Address of current node
L	previous element

Returns

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

5.37.2.10 AK_index_table_exist()

```
int AK_index_table_exist (
            char * indexTblName )
```

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

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

tblName	table name

Returns

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

5.37.2.11 AK_index_test()

```
void AK_index_test ( )
```

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

Author

Lovro Predovan

Returns

No return value

5.37.2.12 AK_InitializelistAd()

```
void AK_InitializelistAd (
            list_ad * L )
```

Function that initialises a linked list.

Author

Unknown

Parameters

list_ad	∗L linked list head

Returns

No return value

5.37.2.13 AK_Insert_NewelementAd()

```
void AK_Insert_NewelementAd (
            int addBlock,
            int indexTd,
            char * attName,
            element_ad elementBefore )
```

Function that inserts a new element into a linked list.

Author

Unknown

Parameters

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

Returns

No return value

5.37.2.14 AK_num_index_attr()

```
int AK_num_index_attr (
            char * indexTblName )
```

Function that fetches the number of elements in a index table.

Author

Lovro Predovan

Parameters

index	table name

Returns

No return value

5.37.2.15 AK_print_index_table()

```
void AK_print_index_table (
            char * indexTblName )
```

Function that prints out the index table.

Author

Matija Šestak, modified for indexes by Lovro Predovan

Parameters

∗*tblName*	table name

Returns

No return value

5.38 file/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 that returns the current value of the sequence.
- int AK_sequence_next_value (char *name)

  Function that 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 that fetches sequence id.
- int AK_sequence_rename (char *old_name, char *new_name)

  Function that 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 a sequence.
- TestResult AK_sequence_test ()

  Function used for sequences testing.

5.38.1 Detailed Description

Provides functions for sequences

5.38.2 Function Documentation

5.38.2.1 AK_sequence_add()

```
int AK_sequence_add (
            char * name,
            int start_value,
            int increment,
            int max_value,
            int min_value,
            int cycle )
```

Function for adding sequence.

Author

Boris Kišić

Parameters

name	name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

sequence_id or EXIT_ERROR

5.38.2.2 AK_sequence_current_value()

```
int AK_sequence_current_value (
            char * name )
```

Function that returns the current value of the sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

current_value or EXIT_ERROR

5.38.2.3 AK_sequence_get_id()

```
int AK_sequence_get_id (
            char * name )
```

Function that fetches sequence id.

Author

Ljubo Barać

Parameters

name	Name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.38.2.4 AK_sequence_modify()

```
int AK_sequence_modify (
             char * name,
             int start_value,
             int increment,
             int max_value,
             int min_value,
             int cycle )
```

Function for modifying a sequence.

Author

Boris Kišić fixed by Ljubo Barać

Parameters

name	Name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.38.2.5 AK_sequence_next_value()

```
int AK_sequence_next_value (
             char * name )
```

Function that returns the next value of the sequence and writes it in a system table as current value.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

next_value or EXIT_ERROR

5.38.2.6 AK_sequence_remove()

```
int AK_sequence_remove (
            char * name )
```

Function for removing sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.38.2.7 AK_sequence_rename()

```
int AK_sequence_rename (
            char * old_name,
            char * new_name )
```

Function that renames the sequence.

Author

Boris Kišić

Parameters

old_name	Name of the sequence to be renamed
new_name	New name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.38.2.8 AK_sequence_test()

```
TestResult AK_sequence_test ( )
```

Function used for sequences testing.

Author

Boris Kišić fixed by Ljubo Barać

Returns

No return value

5.39 file/sequence.h File Reference

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

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

Functions

- int AK_sequence_add (char *name, int start_value, int increment, int max_value, int min_value, int cycle)
  Function for adding sequence.
- int AK_sequence_remove (char *name)
  Function for removing sequence.
- int AK_sequence_current_value (char *name)
  Function that returns the current value of the sequence.
- int AK_sequence_next_value (char *name)
  Function that 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 that 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 a sequence.
- int AK_sequence_get_id (char *name)
  Function that fetches sequence id.
- TestResult AK_sequence_test ()
  Function used for sequences testing.

5.39.1 Detailed Description

Header file that provides functions and defines for sequences

5.39.2 Function Documentation

5.39.2.1 AK_sequence_add()

```
int AK_sequence_add (
            char * name,
            int start_value,
            int increment,
            int max_value,
            int min_value,
            int cycle )
```

Function for adding sequence.

Author

Boris Kišić

Parameters

name	name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

sequence_id or EXIT_ERROR

5.39.2.2 AK_sequence_current_value()

```
int AK_sequence_current_value (
            char * name )
```

Function that returns the current value of the sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

current_value or EXIT_ERROR

5.39.2.3 AK_sequence_get_id()

```
int AK_sequence_get_id (
             char * name )
```

Function that fetches sequence id.

Author

Ljubo Barać

Parameters

name	Name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.39.2.4 AK_sequence_modify()

```
int AK_sequence_modify (
             char * name,
             int start_value,
             int increment,
             int max_value,
             int min_value,
             int cycle )
```

Function for modifying a sequence.

Author

Boris Kišić fixed by Ljubo Barać

Parameters

name	Name of the sequence
start_value	start value of the sequence
increment	increment of the sequence
max_value	maximium value of the sequence
min_value	minimum value of the sequence
cycle	0:non-cyclic sequence, 1:cyclic sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.39.2.5 AK_sequence_next_value()

```
int AK_sequence_next_value (
            char * name )
```

Function that returns the next value of the sequence and writes it in a system table as current value.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

next_value or EXIT_ERROR

5.39.2.6 AK_sequence_remove()

```
int AK_sequence_remove (
            char * name )
```

Function for removing sequence.

Author

Boris Kišić

Parameters

name	name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.39.2.7 AK_sequence_rename()

```
int AK_sequence_rename (
            char * old_name,
            char * new_name )
```

Function that renames the sequence.

/**

Author

Boris Kišić

Parameters

old_name	Name of the sequence to be renamed
new_name	New name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

Author

Boris Kišić

Parameters

old_name	Name of the sequence to be renamed
new_name	New name of the sequence

Returns

EXIT_SUCCESS or EXIT_ERROR

5.39.2.8 AK_sequence_test()

```
TestResult AK_sequence_test ( )
```

Function used for sequences testing.

Author

Boris Kišić fixed by Ljubo Barać

Returns

No return value

5.40 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)

  Temporary function that creates table, and inserts an entry to the system_relation catalog.
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  Temporary function that creates table, and inserts an entry to the system_relation catalog.
- int AK_num_attr (char *tblName)

  Functions that determines the number of attributes in the table.
- int AK_get_num_records (char *tblName)

  Function that determines the number of rows in the table.
- AK_header * AK_get_header (char *tblName)

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

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

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

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

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

  Function that fetches a 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 that 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 the value of an attribute from the row.
- void AK_print_row_to_file (int col_len[ ], struct list_node *row)

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

  Function that prints a table.
- int AK_table_empty (char *tblName)

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

  Function that fetches an 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 that checks 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)
- TestResult AK_table_test ()

  Function for testing table abstraction.
- TestResult AK_op_rename_test ()

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

5.40.1 Detailed Description

Provides functions for table abstraction

5.40.2 Function Documentation

5.40.2.1 AK_check_tables_scheme()

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

Function that checks if tables have the same relation schema.

Author

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

Parameters

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

Returns

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

5.40.2.2 AK_create_table()

```
void AK_create_table (
            char * tblName,
```

```
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.40.2.3 AK_get_attr_index()

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

Function that fetches zero-based index for atrribute.

Author

Matija Šestak.

Parameters

∗tblName	table name
∗attrName	attribute name

Returns

zero-based index

5.40.2.4 AK_get_attr_name()

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

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

Author

Matija Šestak.

Parameters

∗tblName	table name
index	zero-based index

Returns

attribute name

5.40.2.5 AK_get_column()

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

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

Author

Matija Šestak.

Parameters

num	zero-based column index
∗tblName	table name

Returns

column values list

5.40.2.6 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the table header.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return 0
3. else read the first block
4. allocate array
5. copy table header to the array

Parameters

∗tblName	table name

Returns

array of table header

5.40.2.7 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the number of rows in the table.

Author

Matija Šestak.

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

Parameters

∗tableName	table name

Returns

number of rows in the table

5.40.2.8 AK_get_row()

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

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

Author

Markus Schatten, Matija Šestak.

Parameters

num	zero-based row index
*	tblName table name

Returns

row values list

5.40.2.9 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

Author

Dejan Frankovic

Parameters

**table*	table name

Returns

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

5.40.2.10 AK_get_tuple()

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

Function that fetches a value in some row and column.

Author

Matija Šestak.

Parameters

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

Returns

value in the list

5.40.2.11 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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 EXIT_WARNING
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

5.40.2.12 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

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

Author

Mislav Čakarić, edited by Ljubo Barać

Returns

No return value

5.40.2.13 AK_print_row()

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

Function that prints table row.

Author

Dino Laktašić

Parameters

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

Returns

No return value

5.40.2.14 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

Author

Dino Laktašić.

Parameters

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

Returns

printed row spacer

5.40.2.15 AK_print_row_spacer_to_file()

```
void AK_print_row_spacer_to_file (
            int col_len[],
            int length )
```

Function that prints row spacer update by Luka Rajcevic.

Author

Dino Laktašić.

Parameters

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

Returns

printed row spacer

5.40.2.16 AK_print_row_to_file()

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

Function that prints the table row update by Luka Rajcevic.

Author

Dino Laktašić

Parameters

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

Returns

No return value

5.40.2.17 AK_print_table()

```
void AK_print_table (
            char * tblName )
```

Function for printing table.

Author

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

Parameters

∗*tblName*	table name

Returns

No return value

5.40.2.18 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

Author

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

Parameters

∗*tblName*	table name

Returns

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

5.40.2.19 AK_rename()

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

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

Author

Mislav Čakarić edited by Ljubo Barać

Parameters

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

Returns

EXIT_ERROR or EXIT_SUCCESS

5.40.2.20 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

Author

Matija Šestak.

Parameters

*tblName	table name

Returns

true/false

5.40.2.21 AK_table_exist()

```
int AK_table_exist (
            char * tblName )
```

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

Author

Jurica Hlevnjak

Parameters

tblName	table name

Returns

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

5.40.2.22 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

Author

Unknown

Returns

No return value

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

5.40.2.23 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.40.2.24 AK_tuple_to_string()

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

Function that converts tuple value to string.

Author

Matija Šestak.

Parameters

∗tuple	tuple in the list

Returns

tuple value as a string

5.40.2.25 get_row_attr_data()

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

Function that returns the value of an attribute from the row.

Author

Leon Palaić

Parameters

column	index of column atribute
∗row	list with row elements

Returns

atribute data

5.41 file/table.h File Reference

```
#include "../auxi/test.h"
#include "../mm/memoman.h"
#include "../auxi/mempro.h"
#include <time.h>
#include "../sql/drop.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/sequence.h"
#include "../auxi/constants.h"
#include "test.h"
#include "../dm/dbman.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)

  Temporary function that creates table, and inserts an entry to the system_relation catalog.
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  Temporary function that creates table, and inserts an entry to the system_relation catalog.
- int AK_num_attr (char *tblName)

  Functions that determines the number of attributes in the table.
- int AK_get_num_records (char *tblName)

  Function that determines the number of rows in the table.
- AK_header * AK_get_header (char *tblName)

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

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

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

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

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

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

  Function that prints a table.
- int AK_table_empty (char *tblName)

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

  Function that fetches an 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 that checks if tables have the same relation schema.
- char * get_row_attr_data (int column, struct list_node *node)

  Function that returns the value of an attribute from the row.

- TestResult 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)
- TestResult AK_op_rename_test ()

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

5.41.1 Detailed Description

Header file that provides data structures, functions and defines 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

5.41.2 Function Documentation

5.41.2.1 AK_check_tables_scheme()

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

Function that checks if tables have the same relation schema.

Author

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

Parameters

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

Returns

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

5.41.2.2 AK_create_table()

```
void AK_create_table (
            char * tblName,
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.41.2.3 AK_get_attr_index()

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

Function that fetches zero-based index for atrribute.

Author

Matija Šestak.

Parameters

∗tblName	table name
∗attrName	attribute name

Returns

zero-based index

5.41.2.4 AK_get_attr_name()

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

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

Author

Matija Šestak.

Parameters

**tblName*	table name
index	zero-based index

Returns

attribute name

5.41.2.5 AK_get_column()

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

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

Author

Matija Šestak.

Parameters

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

Returns

column values list

5.41.2.6 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the table header.

Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return 0
3. else read the first block
4. allocate array
5. copy table header to the array

Parameters

∗*tblName*	table name

Returns

array of table header

5.41.2.7 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the number of rows in the table.

Author

Matija Šestak.

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

Parameters

∗*tableName*	table name

Returns

number of rows in the table

5.41.2.8 AK_get_row()

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

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

Author

Markus Schatten, Matija Šestak.

Parameters

num	zero-based row index
*	tblName table name

Returns

row values list

5.41.2.9 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

Author

Dejan Frankovic

Parameters

**table*	table name

Returns

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

5.41.2.10 AK_get_tuple()

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

Function that fetches a value in some row and column.

Author

Matija Šestak.

Parameters

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

Returns

value in the list

5.41.2.11 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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 EXIT_WARNING
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

5.41.2.12 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

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

Author

Mislav Čakarić, edited by Ljubo Barać

Returns

No return value

5.41.2.13 AK_print_row()

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

Function that prints table row.

Author

Dino Laktašić

Parameters

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

Returns

No return value

5.41.2.14 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

Author

Dino Laktašić.

Parameters

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

Returns

printed row spacer

5.41.2.15 AK_print_row_spacer_to_file()

```
void AK_print_row_spacer_to_file (
            int col_len[],
            int length )
```

Function that prints row spacer update by Luka Rajcevic.

Author

Dino Laktašić.

Parameters

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

Returns

printed row spacer

5.41.2.16 AK_print_row_to_file()

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

Function that prints the table row update by Luka Rajcevic.

Author

Dino Laktašić

Parameters

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

Returns

No return value

5.41.2.17 AK_print_table()

```
void AK_print_table (
            char * tblName )
```

Function for printing table.

Author

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

Parameters

*tblName	table name

Returns

No return value

5.41.2.18 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

Author

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

Parameters

*tblName	table name

Returns

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

5.41.2.19 AK_rename()

```
int AK_rename (
            char * old_table_name,
```

```
            char * old_attr,
            char * new_table_name,
            char * new_attr )
```

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

Author

Mislav Čakarić edited by Ljubo Barać

Parameters

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

Returns

EXIT_ERROR or EXIT_SUCCESS

5.41.2.20 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

Author

Matija Šestak.

Parameters

∗*tblName*	table name

Returns

true/false

5.41.2.21 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

Author

Unknown

Returns

No return value

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

5.41.2.22 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

Author

Matija Novak, updated by Dino Laktašić

Parameters

table	table name
header	AK_header of the new table
type_segment	type of the new segment

Returns

No return value

5.41.2.23 AK_tuple_to_string()

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

Function that converts tuple value to string.

Author

Matija Šestak.

Parameters

∗tuple	tuple in the list

Returns

tuple value as a string

5.41.2.24 get_row_attr_data()

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

Function that returns the value of an attribute from the row.

Author

Leon Palaić

Parameters

column	index of column atribute
∗row	list with row elements

Returns

atribute data

5.42 mm/memoman.c File Reference

```
#include "memoman.h"
#include "../dm/dbman.h"
```

Include dependency graph for memoman.c:

Functions

- int AK_cache_block (int num, AK_mem_block ∗mem_block)

  Function that caches a block into the memory.
- int AK_cache_AK_malloc ()

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

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

  Function that finds the available block for result caching in a circular array.
- unsigned long AK_generate_result_id (unsigned char ∗str)

  Function that generates a unique hash identifier for each cached result by using djb2 algorithm.
- void AK_cache_result (char ∗srcTable, AK_block ∗temp_block, AK_header header[ ])

  Function that caches the fetched result block in memory.
- int AK_query_mem_AK_malloc ()

Function that initializes the global query memory (variable query_mem)

- void AK_query_mem_AK_free ()

  Function that releases the global query memory (variable query_mem)

- int AK_memoman_init ()

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

- AK_mem_block * AK_get_block (int num)

  Function that reads a block from the 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.

- int AK_release_oldest_cache_block ()

  Functions that flushes the oldest block to disk and recalculates the next block to remove.

- int AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.

- int AK_refresh_cache ()

  Function that re-reads all the blocks from the disk.

- table_addresses * AK_get_index_segment_addresses (char *segmentName)

  Function for getting a index segment address.

- table_addresses * AK_get_segment_addresses (char *segmentName)

  Function for getting a relation segment address.

- table_addresses * AK_get_segment_addresses_internal (char *tableName, char *segmentName)

  Function for getting addresses of some table.

- int AK_get_system_table_address (const char *name)

  Function that gets the address of a system table by name.

- 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 that finds AK_free space in some block betwen block addresses. It's made for insert_row()

- int AK_init_new_extent (char *table_name, int extent_type)

  Function that extends the segment.

- int AK_flush_cache ()

  Function that flushes memory blocks to disk file.

- TestResult **AK_memoman_test** ()
- TestResult **AK_memoman_test2** ()

5.42.1 Detailed Description

Defines functions for the memory manager of Kalashnikov DB

5.42.2 Function Documentation

5.42.2.1 AK_cache_AK_malloc()

```
int AK_cache_AK_malloc ( )
```

Function that 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

5.42.2.2 AK_cache_block()

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

Function that caches a block into the memory.

Author

Nikola Bakoš, Matija Šestak(revised)

Parameters

num	block number (address)
mem_block	address of memmory block

Returns

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

read the block from the given address

set dirty bit in mem_block struct

get the timestamp

set timestamp_read

set timestamp_last_change

5.42.2.3 AK_cache_result()

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

Function that caches the fetched result block in memory.

Author

Mario Novoselec

5.42.2.4 AK_find_AK_free_space()

```
int AK_find_AK_free_space (
            table_addresses * addresses )
```

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

Author

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

Parameters

address	addresses of extents

Returns

address of the block to write in

5.42.2.5 AK_find_available_result_block()

```
int AK_find_available_result_block ( )
```

Function that finds the available block for result caching in a circular array.

Author

Mario Novoselec

Returns

available_index

5.42.2.6 AK_flush_cache()

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

Author

Matija Šestak, updated by Antonio Martinović

Returns

EXIT_SUCCESS

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

block is clean after successfuly writing it to disk

5.42.2.7 AK_generate_result_id()

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

Function that generates a unique hash identifier for each cached result by using djb2 algorithm.

Author

Mario Novoselec

Returns

hash

5.42.2.8 AK_get_block()

```
AK_mem_block* AK_get_block (
            int num )
```

Function that reads a block from the 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, Antonio Martinović

Parameters

num	block number (address)

Returns

segment start address

found cached! we're done here

while looking for block we also want to find an empty block in case that the actual block is not found then there is no need to run through the blocks twice

created new cache block for specified address

no free cache blocks found, we need to clear some now

no cache for you

5.42.2.9 AK_get_index_addresses()

```
table_addresses* AK_get_index_addresses (
            char * index )
```

Function for getting addresses of some index.

Author

Mislav Čakarić

Parameters

index	index name that you search for

Returns

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

5.42.2.10 AK_get_index_segment_addresses()

```
table_addresses* AK_get_index_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

Parameters

segmentName	table name that you search for

Returns

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

5.42.2.11 AK_get_segment_addresses()

```
table_addresses* AK_get_segment_addresses (
            char * segmentName )
```

Function for getting a relation segment address.

Function for getting a index segment address.

@Author Antonio Martinović

Parameters

segmentName	table name that you search for

Returns

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

5.42.2.12 AK_get_segment_addresses_internal()

```
table_addresses* AK_get_segment_addresses_internal (
            char * tableName,
            char * segmentName )
```

Function for getting addresses of some table.

Author

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

Parameters

tableName	table name that you search for
segmentName	segment name

Returns

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

5.42.2.13 AK_get_system_table_address()

```
int AK_get_system_table_address (
            const char * name )
```

Function that gets the address of a system table by name.

Author

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

Parameters

name	of system table

Returns

table address

5.42.2.14 AK_get_table_addresses()

```
table_addresses* AK_get_table_addresses (
            char * table )
```

Function for getting addresses of some table.

Author

Mislav Čakarić

Parameters

table	table name that you search for

Returns

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

5.42.2.15 AK_init_new_extent()

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

Function that extends the segment.

Author

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

Parameters

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

Returns

address of new extent, otherwise EXIT_ERROR

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

5.42.2.16 AK_mem_block_modify()

```
int AK_mem_block_modify (
            AK_mem_block * mem_block,
            int dirty )
```

Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.

Author

Alen Novosel.

5.42.2.17 AK_memoman_init()

```
int AK_memoman_init ( )
```

Function that initializes the 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

5.42.2.18 AK_query_mem_AK_free()

```
void AK_query_mem_AK_free ( )
```

Function that releases the global query memory (variable query_mem)

Author

Elvis Popović

5.42.2.19 AK_query_mem_AK_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function that 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

5.42.2.20 AK_redo_log_AK_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function that 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

5.42.2.21 AK_refresh_cache()

```
int AK_refresh_cache ( )
```

Function that re-reads all the blocks from the disk.

Author

Matija Šestak.

Returns

EXIT_SUCCESS

5.42.2.22 AK_release_oldest_cache_block()

```
int AK_release_oldest_cache_block ( )
```

Functions that flushes the oldest block to disk and recalculates the next block to remove.

Author

Antonio Martinović

Returns

index of flushed cache block

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

block is clean after successfuly writing it to disk

5.43 mm/memoman.h File Reference

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

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

Classes

- struct AK_mem_block

  Structure that defines a block of data in memory.
- struct AK_db_cache

  Structure that defines global cache memory.
- struct AK_command_recovery_struct

  recovery structure used to recover commands from binary file
- struct AK_redo_log

  Structure that defines global redo log.
- struct AK_query_mem_lib

  Structure that defines global query memory for libraries.
- struct AK_query_mem_dict

  Structure that defines global query memory for data dictionaries.
- struct AK_results

  Structure used for in-memory result caching.
- struct AK_query_mem_result

  Structure that defines global query memory for results.
- struct AK_query_mem

  Structure that defines global query memory.

Functions

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

  Function that caches the fetched result block in memory.
- int AK_find_available_result_block ()

  Function that finds the available block for result caching in a circular array.
- unsigned long AK_generate_result_id (unsigned char *str)

  Function that generates a unique hash identifier for each cached result by using djb2 algorithm.
- int AK_cache_block (int num, AK_mem_block *mem_block)

  Function that caches a block into the memory.
- int AK_cache_AK_malloc ()

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

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

  Function that initializes the global query memory (variable query_mem)
- void AK_query_mem_AK_free ()

  Function that releases the global query memory (variable query_mem)
- int AK_memoman_init ()

  Function that initializes the memory manager (cache, redo log and query memory)
- AK_mem_block * AK_get_block (int num)

  Function that reads a block from the 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.
- int AK_release_oldest_cache_block ()

  Functions that flushes the oldest block to disk and recalculates the next block to remove.
- int AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.
- int AK_refresh_cache ()

  Function that re-reads all the blocks from the disk.

- table_addresses * AK_get_segment_addresses_internal (char *tableName, char *segmentName)
  Function for getting addresses of some table.
- table_addresses * AK_get_segment_addresses (char *segmentName)
  Function for getting a index segment address.
- table_addresses * AK_get_index_segment_addresses (char *segmentName)
  Function for getting a index segment address.
- 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 that finds AK_free space in some block betwen block addresses. It's made for insert_row()
- int AK_init_new_extent (char *table_name, int extent_type)
  Function that extends the segment.
- int AK_flush_cache ()
  Function that flushes memory blocks to disk file.
- TestResult **AK_memoman_test** ()
- TestResult **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.

5.43.1 Detailed Description

Header file that contains data structures, defines and functions for the memory manager of Kalashnikov DB

5.43.2 Function Documentation

5.43.2.1 AK_cache_AK_malloc()

```
int AK_cache_AK_malloc ( )
```

Function that 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

5.43.2.2 AK_cache_block()

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

Function that caches a block into the memory.

Author

Nikola Bakoš, Matija Šestak(revised)

Parameters

num	block number (address)
mem_block	address of memmory block

Returns

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

read the block from the given address

set dirty bit in mem_block struct

get the timestamp

set timestamp_read

set timestamp_last_change

5.43.2.3 AK_cache_result()

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

Function that caches the fetched result block in memory.

Author

Mario Novoselec

5.43.2.4 AK_find_AK_free_space()

```
int AK_find_AK_free_space (
            table_addresses * addresses )
```

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

Author

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

Parameters

address	addresses of extents

Returns

address of the block to write in

5.43.2.5 AK_find_available_result_block()

```
int AK_find_available_result_block ( )
```

Function that finds the available block for result caching in a circular array.

Author

Mario Novoselec

Returns

available_index

5.43.2.6 AK_flush_cache()

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

Author

Matija Šestak, updated by Antonio Martinović

Returns

EXIT_SUCCESS

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

block is clean after successfuly writing it to disk

5.43.2.7 AK_generate_result_id()

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

Function that generates a unique hash identifier for each cached result by using djb2 algorithm.

Author

Mario Novoselec

Returns

hash

5.43.2.8 AK_get_block()

```
AK_mem_block* AK_get_block (
            int num )
```

Function that reads a block from the 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, Antonio Martinović

Parameters

num	block number (address)

Returns

segment start address

found cached! we're done here

while looking for block we also want to find an empty block in case that the actual block is not found then there is no need to run through the blocks twice

created new cache block for specified address

no free cache blocks found, we need to clear some now

no cache for you

5.43.2.9 AK_get_index_addresses()

```
table_addresses* AK_get_index_addresses (
            char * index )
```

Function for getting addresses of some index.

Author

Mislav Čakarić

Parameters

index	index name that you search for

Returns

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

5.43.2.10 AK_get_index_segment_addresses()

```
table_addresses* AK_get_index_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

Parameters

segmentName	table name that you search for

Returns

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

5.43.2.11 AK_get_segment_addresses()

```
table_addresses* AK_get_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

Parameters

segmentName	table name that you search for

Returns

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

Function for getting a index segment address.

@Author Antonio Martinović

Parameters

segmentName	table name that you search for

Returns

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

5.43.2.12 AK_get_segment_addresses_internal()

```
table_addresses* AK_get_segment_addresses_internal (
            char * tableName,
            char * segmentName )
```

Function for getting addresses of some table.

Author

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

Parameters

tableName	table name that you search for
segmentName	segment name

Returns

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

5.43.2.13 AK_get_table_addresses()

```
table_addresses* AK_get_table_addresses (
            char * table )
```

Function for getting addresses of some table.

Author

Mislav Čakarić

Parameters

table	table name that you search for

Returns

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

5.43.2.14 AK_init_new_extent()

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

Function that extends the segment.

Author

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

Parameters

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

Returns

address of new extent, otherwise EXIT_ERROR

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

5.43.2.15 AK_mem_block_modify()

```
int AK_mem_block_modify (
            AK_mem_block * mem_block,
            int dirty )
```

Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.

Author

Alen Novosel.

5.43.2.16 AK_memoman_init()

```
int AK_memoman_init ( )
```

Function that initializes the 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

5.43.2.17 AK_query_mem_AK_free()

```
void AK_query_mem_AK_free ( )
```

Function that releases the global query memory (variable query_mem)

Author

Elvis Popović

5.43.2.18 AK_query_mem_AK_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function that 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

5.43.2.19 AK_redo_log_AK_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function that 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

5.43.2.20 AK_refresh_cache()

```
int AK_refresh_cache ( )
```

Function that re-reads all the blocks from the disk.

Author

Matija Šestak.

Returns

EXIT_SUCCESS

5.43.2.21 AK_release_oldest_cache_block()

```
int AK_release_oldest_cache_block ( )
```

Functions that flushes the oldest block to disk and recalculates the next block to remove.

Author

Antonio Martinović

Returns

index of flushed cache block

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

block is clean after successfuly writing it to disk

5.44 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)

  Function that prints optimization table for testing purposes.
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  Function that calls and executes 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)

  Function that executes 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.
- TestResult AK_query_optimization_test ()

Variables

- int **error_message** =0

5.44.1 Detailed Description

Provides functions for general query optimization

5.44.2 Function Documentation

5.44.2.1 AK_execute_rel_eq()

```
struct list_node* AK_execute_rel_eq (
            struct list_node * list_query,
            const char rel_eq,
            const char * FLAGS )
```

Function that calls and executes relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection

Author

Dino Laktašić.

Parameters

∗list_query	RA expresion list where we need to apply relational equivalences rules
rel_eq	rel_eq to execute
∗FLAGS	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns struct list_node (RA expresion list) optimized by given relational equivalence rule

5.44.2.2 AK_print_optimized_query()

```
void AK_print_optimized_query (
            struct list_node * list_query )
```

Function that prints optimization table for testing purposes.

Author

Dino Laktašić.

Parameters

∗list_query	optimized RA expresion list

Returns

list output

5.44.2.3 AK_query_optimization()

```
struct list_node* AK_query_optimization (
            struct list_node * list_query,
            const char * FLAGS,
            const int DIFF_PLANS )
```

Function that executes all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

Author

Dino Laktašić.

Parameters

∗list_query	RA expresion list where we need to apply relational equivalences rules
∗FLAGS	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns AK_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

5.44.2.4 AK_query_optimization_test()

```
TestResult AK_query_optimization_test ( )
```

Author

Dino Laktašić

Parameters

Function	for testing ∗list_query query to be optimized

Returns

No return value

5.45 opti/query_optimization.h File Reference

```
#include "../auxi/test.h"
#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)

  Function that prints optimization table for testing purposes.
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  Function that calls and executes 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)

  Function that executes 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.
- TestResult AK_query_optimization_test ()

5.45.1 Detailed Description

Header file that provides data structure, functions and defines for general query optimization

5.45.2 Function Documentation

5.45.2.1 AK_execute_rel_eq()

```
struct list_node* AK_execute_rel_eq (
            struct list_node * list_query,
            const char rel_eq,
            const char * FLAGS )
```

Function that calls and executes relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection

Author

Dino Laktašić.

Parameters

∗list_query	RA expresion list where we need to apply relational equivalences rules
rel_eq	rel_eq to execute
∗FLAGS	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns struct list_node (RA expresion list) optimized by given relational equivalence rule

5.45.2.2 AK_print_optimized_query()

```
void AK_print_optimized_query (
            struct list_node * list_query )
```

Function that prints optimization table for testing purposes.

Author

Dino Laktašić.

Parameters

∗list_query	optimized RA expresion list

Returns

list output

5.45.2.3 AK_query_optimization()

```
struct list_node* AK_query_optimization (
            struct list_node * list_query,
            const char * FLAGS,
            const int DIFF_PLANS )
```

Function that executes all relational equivalences provided by FLAGS (one or more), if DIFF_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

Author

Dino Laktašić.

Parameters

∗list_query	RA expresion list where we need to apply relational equivalences rules
∗FLAGS	flags for relation equivalences (execute rel_eq for given flags)

Returns

returns AK_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

5.45.2.4 AK_query_optimization_test()

```
TestResult AK_query_optimization_test ( )
```

Author

Dino Laktašić

Parameters

Function	for testing ∗list_query query to be optimized

Returns

No return value

5.46 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 generation of 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.
- TestResult AK_rel_eq_assoc_test ()

  Function for testing relational equivalences regarding associativity.

5.46.1 Detailed Description

Provides functions for relational equivalences regarding associativity

5.46.2 Function Documentation

5.46.2.1 AK_compare()

```
int AK_compare (
            const void * a,
            const void * b )
```

Function for Struct cost_eval comparison.

Author

Dino Laktašić

Parameters

*a	first value
*b	second value

Returns

returns result of comparison

5.46.2.2 AK_print_rel_eq_assoc()

```
void AK_print_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Function for printing RA expresion struct list_node.

Author

Dino Laktašić.

Parameters

*list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.46.2.3 AK_rel_eq_assoc()

```
struct list_node* AK_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Main function for generation of RA expresion according to associativity equivalence rules.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.46.2.4 AK_rel_eq_assoc_test()

```
TestResult AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

Author

Dino Laktašić.

Returns

No return value

5.47 opti/rel_eq_assoc.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
```

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

Classes

- struct cost_eval_t

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

Typedefs

- typedef struct cost_eval_t **cost_eval**

Functions

- int AK_compare (const void ∗a, const void ∗b)

  Function for Struct cost_eval comparison.
- struct list_node ∗ AK_rel_eq_assoc (struct list_node ∗list_rel_eq)

  Main function for generation of 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.
- TestResult AK_rel_eq_assoc_test ()

  Function for testing relational equivalences regarding associativity.

5.47.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences regarding associativity

5.47.2 Function Documentation

5.47.2.1 AK_compare()

```
int AK_compare (
            const void * a,
            const void * b )
```

Function for Struct cost_eval comparison.

Author

Dino Laktašić

Parameters

∗*a*	first value
∗*b*	second value

Returns

returns result of comparison

5.47.2.2 AK_print_rel_eq_assoc()

```
void AK_print_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Function for printing RA expresion struct list_node.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.47.2.3 AK_rel_eq_assoc()

```
struct list_node* AK_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Main function for generation of RA expresion according to associativity equivalence rules.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.47.2.4 AK_rel_eq_assoc_test()

```
TestResult AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

Author

Dino Laktašić.

Returns

No return value

5.48 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)
  Function that checks if the selection can commute with theta-join or product.
- TestResult AK_rel_eq_comut_test ()
  Function that tests relational equivalences regarding commutativity.

5.48.1 Detailed Description

Provides functions for relational equivalences regarding commutativity

5.48.2 Function Documentation

5.48.2.1 AK_print_rel_eq_comut()

```
void AK_print_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Function for printing optimized relation equivalence expression list regarding commutativity.

Author

Davor Tomala

Parameters

∗list_rel_eq	RA expresion as the struct list_node

5.48.2.2 AK_rel_eq_commute_with_theta_join()

```
char* AK_rel_eq_commute_with_theta_join (
            char * cond,
            char * tblName )
```

Function that checks if the selection can commute with theta-join or product.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

Parameters

∗cond	condition array that contains condition data
∗tblName	name of the table

Returns

pointer to char array that contains new condition for a given table

5.48.2.3 AK_rel_eq_comut()

```
struct list_node* AK_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to commutativity equivalence rules.

Author

Davor Tomala

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.48.2.4 AK_rel_eq_comut_test()

```
TestResult AK_rel_eq_comut_test ( )
```

Function that tests relational equivalences regarding commutativity.

Author

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

Returns

No return vlaue

5.49 opti/rel_eq_comut.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "./rel_eq_selection.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
```

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

Functions

- void AK_print_rel_eq_comut (struct list_node ∗list_rel_eq)

  Function for printing optimized relation equivalence expression list regarding commutativity.
- struct list_node ∗ AK_rel_eq_comut (struct list_node ∗list_rel_eq)

  Main function for generating RA expresion according to commutativity equivalence rules.
- char ∗ AK_rel_eq_commute_with_theta_join (char ∗cond, char ∗tblName)

  Function that checks if the selection can commute with theta-join or product.
- TestResult AK_rel_eq_comut_test ()

  Function that tests relational equivalences regarding commutativity.

5.49.1 Detailed Description

Header file that provides data structures,functions and defines for relational equivalences regarding comutativity

5.49.2 Function Documentation

5.49.2.1 AK_print_rel_eq_comut()

```
void AK_print_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Function for printing optimized relation equivalence expression list regarding commutativity.

Author

Davor Tomala

Parameters

∗list_rel_eq	RA expresion as the struct list_node

5.49.2.2 AK_rel_eq_commute_with_theta_join()

```
char* AK_rel_eq_commute_with_theta_join (
            char * cond,
            char * tblName )
```

Function that checks if the selection can commute with theta-join or product.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

Parameters

∗cond	condition array that contains condition data
∗tblName	name of the table

Returns

pointer to char array that contains new condition for a given table

5.49.2.3 AK_rel_eq_comut()

```
struct list_node* AK_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to commutativity equivalence rules.

Author

Davor Tomala

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.49.2.4 AK_rel_eq_comut_test()

```
TestResult AK_rel_eq_comut_test ( )
```

Function that tests relational equivalences regarding commutativity.

Author

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

Returns

No return vlaue

5.50 opti/rel_eq_projection.c File Reference

```
#include "rel_eq_projection.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_projection.c:

Functions

- int AK_rel_eq_is_subset (struct list_node *list_elem_set, struct list_node *list_elem_subset)

  Function that checks 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)

  Function that checks if selection uses only attributes retained by the projection before commuting.
- struct list_node * AK_rel_eq_get_attributes (char *tblName)

  Function that gets attributes for a given table and store them to the struct list_node.
- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)

  Function used for 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)

  Function used for 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.
- TestResult AK_rel_eq_projection_test ()

  Function for testing rel_eq_selection.

5.50.1 Detailed Description

Provides functions for for relational equivalences in projection

5.50.2 Function Documentation

5.50.2.1 AK_print_rel_eq_projection()

```
void AK_print_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Function for printing AK_list to the screen.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the AK_list

Returns

No return value

5.50.2.2 AK_rel_eq_can_commute()

```
int AK_rel_eq_can_commute (
            struct list_node * list_elem_attribs,
            struct list_node * list_elem_conds )
```

Function that checks if selection uses only attributes retained by the projection before commuting.

Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break
4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT_SUCCESS, return EXIT_FAILURE

Parameters

list_elem_attribs	list element containing projection data
list_elem_conds	list element containing selection condition data

Returns

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

5.50.2.3 AK_rel_eq_collect_cond_attributes()

```
char* AK_rel_eq_collect_cond_attributes (
            struct list_node * list_elem )
```

Function used for filtering and returning only attributes from selection or theta_join condition.

Author

Dino Laktašić.

Parameters

list_elem	list element that contains selection or theta_join condition data

Returns

only attributes from selection or theta_join condition as the AK_list

5.50.2.4 AK_rel_eq_get_attributes()

```
struct list_node* AK_rel_eq_get_attributes (
            char * tblName )
```

Function that gets attributes for a given table and store them to the struct list_node.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct list_node
4. For each attribute in table header, insert attribute in struct list_node as new struct list_node element
5. return struct list_node

Parameters

∗*tblName*	name of the table

Returns

struct list_node

5.50.2.5 AK_rel_eq_is_subset()

```
int AK_rel_eq_is_subset (
            struct list_node * list_elem_set,
            struct list_node * list_elem_subset )
```

Function that checks 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 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (p[L](R2))

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

list_elem_set	first list element containing projection attributes
list_elem_subset	second list element containing projection attributes

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.50.2.6 AK_rel_eq_projection()

```
struct list_node* AK_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to projection equivalence rules.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the AK_list

Returns

optimised RA expresion as the AK_list

5.50.2.7 AK_rel_eq_projection_attributes()

```
char* AK_rel_eq_projection_attributes (
            char * attribs,
            char * tblName )
```

Function used for filtering and returning only those attributes from list of projection attributes that exist in the given table

Author

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK_list
4. if exists append attribute to the dynamic atributes char array
5. return pointer to char array with stored attribute/s

Parameters

∗attribs	projection attributes delimited by ";" (ATTR_DELIMITER)
∗tblName	name of the table

Returns

filtered list of projection attributes as the AK_list

5.50.2.8 AK_rel_eq_projection_test()

```
TestResult AK_rel_eq_projection_test ( )
```

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.50.2.9 AK_rel_eq_remove_duplicates()

```
char* AK_rel_eq_remove_duplicates (
            char * attribs )
```

Function which removes duplicate attributes from attributes expresion.

Author

Dino Laktašić.

Parameters

*attribs	attributes from which to remove duplicates

Returns

pointer to char array without duplicate attributes

5.51 opti/rel_eq_projection.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../auxi/mempro.h"
```

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

Functions

- int AK_rel_eq_is_subset (struct list_node *list_elem_set, struct list_node *list_elem_subset)

  Function that checks 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)

  Function that checks if selection uses only attributes retained by the projection before commuting.
- struct list_node * AK_rel_eq_get_attributes (char *tblName)

  Function that gets attributes for a given table and store them to the struct list_node.
- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)

  Function used for 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)

  Function used for 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.
- TestResult AK_rel_eq_projection_test ()

  Function for testing rel_eq_selection.

5.51.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences in projection

5.51.2 Function Documentation

5.51.2.1 AK_print_rel_eq_projection()

```
void AK_print_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Function for printing AK_list to the screen.

Author

Dino Laktašić.

Parameters

**list_rel_eq*	RA expresion as the AK_list

Returns

No return value

5.51.2.2 AK_rel_eq_can_commute()

```
int AK_rel_eq_can_commute (
            struct list_node * list_elem_attribs,
            struct list_node * list_elem_conds )
```

Function that checks if selection uses only attributes retained by the projection before commuting.

Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break
4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT_SUCCESS, return EXIT_FAILURE

Parameters

list_elem_attribs	list element containing projection data
list_elem_conds	list element containing selection condition data

Returns

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

5.51.2.3 AK_rel_eq_collect_cond_attributes()

```
char* AK_rel_eq_collect_cond_attributes (
            struct list_node * list_elem )
```

Function used for filtering and returning only attributes from selection or theta_join condition.

Author

Dino Laktašić.

Parameters

list_elem	list element that contains selection or theta_join condition data

Returns

only attributes from selection or theta_join condition as the AK_list

5.51.2.4 AK_rel_eq_get_attributes()

```
struct list_node* AK_rel_eq_get_attributes (
            char * tblName )
```

Function that gets attributes for a given table and store them to the struct list_node.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct list_node
4. For each attribute in table header, insert attribute in struct list_node as new struct list_node element
5. return struct list_node

Parameters

**tblName*	name of the table

Returns

struct list_node

5.51.2.5 AK_rel_eq_is_subset()

```
int AK_rel_eq_is_subset (
            struct list_node * list_elem_set,
            struct list_node * list_elem_subset )
```

Function that checks if some set of attributes is subset of larger set, used in cascading of the projections.

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

list_elem_set	first list element containing projection attributes
list_elem_subset	second list element containing projection attributes

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

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 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (p[L](R2))

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

list_elem_set	first list element containing projection attributes
list_elem_subset	second list element containing projection attributes

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.51.2.6 AK_rel_eq_projection()

```
struct list_node* AK_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to projection equivalence rules.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the AK_list

Returns

optimised RA expresion as the AK_list

5.51.2.7 AK_rel_eq_projection_attributes()

```
char* AK_rel_eq_projection_attributes (
            char * attribs,
            char * tblName )
```

Function used for filtering and returning only those attributes from list of projection attributes that exist in the given table

Author

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK_list
4. if exists append attribute to the dynamic atributes char array
5. return pointer to char array with stored attribute/s

Parameters

∗attribs	projection attributes delimited by ";" (ATTR_DELIMITER)
∗tblName	name of the table

Returns

filtered list of projection attributes as the AK_list

5.51.2.8 AK_rel_eq_projection_test()

```
TestResult AK_rel_eq_projection_test ( )
```

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.51.2.9 AK_rel_eq_remove_duplicates()

```
char* AK_rel_eq_remove_duplicates (
            char * attribs )
```

Function which removes duplicate attributes from attributes expresion.

Author

Dino Laktašić.

Parameters

*attribs	attributes from which to remove duplicates

Returns

pointer to char array without duplicate attributes

5.52 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)

  Function that checks if some set of attributes is subset of larger set.
- char * AK_rel_eq_get_atrributes_char (char *tblName)

  Function that fetches 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)

  Function that checks if two sets share one or more of it's attributes.
- struct list_node * AK_rel_eq_split_condition (char *cond)

  Function that checks 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.
- TestResult AK_rel_eq_selection_test ()

  Function for testing rel_eq_selection.

5.52.1 Detailed Description

Provides functions for for relational equivalences in selection

5.52.2 Function Documentation

5.52.2.1 AK_print_rel_eq_selection()

```
void AK_print_rel_eq_selection (
            struct list_node * list_rel_eq )
```

Function for printing struct list_node to the screen.

Author

Dino Laktašić.

Parameters

*list_rel_eq	RA expresion as the struct list_node

Returns

void

5.52.2.2 AK_rel_eq_cond_attributes()

```
char* AK_rel_eq_cond_attributes (
            char * cond )
```

Function for filtering and returning attributes from condition.

Author

Dino Laktašić.

Parameters

∗cond	condition array that contains condition data

Returns

pointer to array that contains attributes for a given condition

5.52.2.3 AK_rel_eq_get_atrributes_char()

```
char* AK_rel_eq_get_atrributes_char (
            char * tblName )
```

Function that fetches attributes for a given table and store them to the char array.

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct list_node
5. For each attribute in table header, insert attribute in the array
6. Delimit each new attribute with ";" (ATTR_DELIMITER)
7. return pointer to char array

Parameters

∗tblName	name of the table

Returns

pointer to char array

5.52.2.4 AK_rel_eq_is_attr_subset()

```
int AK_rel_eq_is_attr_subset (
            char * set,
            char * subset )
```

Function that checks if some set of attributes is subset of larger set.

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

∗*set*	set array
∗*subset*	subset array

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.52.2.5 AK_rel_eq_selection()

```
struct list_node* AK_rel_eq_selection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to selection equivalence rules.

Author

Dino Laktašić.

Parameters

∗*list_rel_eq*	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.52.2.6 AK_rel_eq_selection_test()

```
TestResult AK_rel_eq_selection_test ( )
```

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.52.2.7 AK_rel_eq_share_attributes()

```
int AK_rel_eq_share_attributes (
            char * set,
            char * subset )
```

Function that checks if two sets share one or more of it's attributes.

Author

Dino Laktašić.

1. If is empty set or subset returns EXIT_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT_SUCCESS
4. else remove unused pointers and return EXIT_FAILURE

Parameters

∗set	first set of attributes delimited by ";" (ATTR_DELIMITER)
∗subset	second set of attributes delimited by ";" (ATTR_DELIMITER)

Returns

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

5.52.2.8 AK_rel_eq_split_condition()

```
struct list_node* AK_rel_eq_split_condition (
            char * cond )
```

Function that checks 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)

Break conjunctive conditions to individual conditions.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition
5. else if token equals to "AND" and id equals to 0 discarge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

Parameters

∗*cond*	condition array that contains condition data
∗*tblName*	name of the table

Returns

pointer to char array that contains new condition for a given table

Author

Dino Laktašić.

Break conjunctive conditions to individual conditions (currently not used - commented in main AK_rel_eq_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct list_node
2. Remove unused pointers and return the conditions list

Parameters

∗*cond*	condition expression

Returns

conditions list

5.53 opti/rel_eq_selection.h File Reference

```
#include "../auxi/test.h"
#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)

  Function that checks if some set of attributes is subset of larger set.
- char * AK_rel_eq_get_atrributes_char (char *tblName)

  Function that fetches 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)

  Function that checks if two sets share one or more of it's attributes.
- struct list_node * AK_rel_eq_split_condition (char *cond)

  Break conjunctive conditions to individual conditions.
- 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.
- TestResult AK_rel_eq_selection_test ()

  Function for testing rel_eq_selection.

5.53.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences in selection

5.53.2 Function Documentation

5.53.2.1 AK_print_rel_eq_selection()

```
void AK_print_rel_eq_selection (
            struct list_node * list_rel_eq )
```

Function for printing struct list_node to the screen.

Author

Dino Laktašić.

Parameters

*list_rel_eq	RA expresion as the struct list_node

Returns

void

5.53.2.2 AK_rel_eq_cond_attributes()

```
char* AK_rel_eq_cond_attributes (
            char * cond )
```

Function for filtering and returning attributes from condition.

Author

Dino Laktašić.

Parameters

∗cond	condition array that contains condition data

Returns

pointer to array that contains attributes for a given condition

5.53.2.3 AK_rel_eq_get_atrributes_char()

```
char* AK_rel_eq_get_atrributes_char (
            char * tblName )
```

Function that fetches attributes for a given table and store them to the char array.

Author

Dino Laktašić.

Parameters

∗tblName	name of the table

Returns

pointer to char array

Author

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct list_node
5. For each attribute in table header, insert attribute in the array

6. Delimit each new attribute with ";" (ATTR_DELIMITER)
7. return pointer to char array

Parameters

∗*tblName*	name of the table

Returns

pointer to char array

5.53.2.4 AK_rel_eq_is_attr_subset()

```
int AK_rel_eq_is_attr_subset (
            char * set,
            char * subset )
```

Function that checks if some set of attributes is subset of larger set.

Author

Dino Laktašić.

Parameters

∗*set*	set array
∗*subset*	subset array

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT_SUCCESS

Parameters

∗*set*	set array
∗*subset*	subset array

Returns

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

5.53.2.5 AK_rel_eq_selection()

```
struct list_node* AK_rel_eq_selection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to selection equivalence rules.

Author

Dino Laktašić.

Parameters

∗list_rel_eq	RA expresion as the struct list_node

Returns

optimised RA expresion as the struct list_node

5.53.2.6 AK_rel_eq_selection_test()

```
TestResult AK_rel_eq_selection_test ( )
```

Function for testing rel_eq_selection.

Author

Dino Laktašić.

Returns

No return value

5.53.2.7 AK_rel_eq_share_attributes()

```
int AK_rel_eq_share_attributes (
            char * set,
            char * subset )
```

Function that checks if two sets share one or more of it's attributes.

Author

Dino Laktašić.

Parameters

∗set	first set of attributes delimited by ";" (ATTR_DELIMITER)
∗subset	second set of attributes delimited by ";" (ATTR_DELIMITER)

Returns

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

Author

Dino Laktašić.

1. If is empty set or subset returns EXIT_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT_SUCCESS
4. else remove unused pointers and return EXIT_FAILURE

Parameters

∗set	first set of attributes delimited by ";" (ATTR_DELIMITER)
∗subset	second set of attributes delimited by ";" (ATTR_DELIMITER)

Returns

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

5.53.2.8 AK_rel_eq_split_condition()

```
struct list_node* AK_rel_eq_split_condition (
            char * cond )
```

Break conjunctive conditions to individual conditions.

Author

Dino Laktašić.

Parameters

∗cond	condition expression

Returns

conditions list

Break conjunctive conditions to individual conditions.

Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition
5. else if token equals to "AND" and id equals to 0 discarge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

Parameters

∗*cond*	condition array that contains condition data
∗*tblName*	name of the table

Returns

pointer to char array that contains new condition for a given table

Author

Dino Laktašić.

Break conjunctive conditions to individual conditions (currently not used - commented in main AK_rel_eq_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct list_node
2. Remove unused pointers and return the conditions list

Parameters

∗*cond*	condition expression

Returns

conditions list

5.54 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.
- char * AK_get_timestamp ()

  Function that returns the current timestamp.

5.54.1 Detailed Description

Header file that provides functions and defines for archive logging

5.54.2 Function Documentation

5.54.2.1 AK_archive_log()

```
void AK_archive_log (
            int sig )
```

Function for making archive log.

Author

Dražen Bandić, update by Tomislav Turek

Returns

No retun value

Function that creates a binary file that stores all commands that failed to execute with a number that shows the size of how many commands failed.

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

Author

Dražen Bandić, update by Tomislav Turek

Returns

No retun value

5.54.2.2 AK_get_timestamp()

```
char* AK_get_timestamp ( )
```

Function that returns the current timestamp.

Author

Dražen Bandić main logic, replaced by Tomislav Turek

Returns

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

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.bin

5.55 rec/recovery.c File Reference

```
#include "recovery.h"
```

Include dependency graph for recovery.c:

Functions

- void AK_recover_archive_log (char *fileName)

  Function that reads the binary file in which last commands were saved, and executes them.
- void AK_recovery_insert_row (char *table, int commandNumber)

  Function that inserts a new row in the table with attributes.
- int recovery_insert_row (char *table, char **attr_name, char **attributes, int n, int *type)

  Function that inserts row in table.
- char ** AK_recovery_tokenize (char *input, char *delimiter, int valuesOrNot)

  Function that 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.
- TestResult AK_recovery_test ()

  Function for recovery testing.
- void AK_load_chosen_log ()

  Executes the recovery operation for the chosen bin file.
- void AK_load_latest_log ()

  Executes the recovery operation for the latest bin file.

Variables

- short grandfailure = 0

5.55.1 Detailed Description

Provides recovery functions.

5.55.2 Function Documentation

5.55.2.1 AK_load_chosen_log()

```
void AK_load_chosen_log ( )
```

Executes the recovery operation for the chosen bin file.

Function lists the contents of the archive_log directory. The user then types in the name of the desired bin file to open and perform the neccessary actions.

Author

Matija Večenaj

Parameters

none	

Returns

no value

5.55.2.2 AK_load_latest_log()

```
void AK_load_latest_log ( )
```

Executes the recovery operation for the latest bin file.

Function reads the latest.txt file which contains the name of the latest bin file that's been created. Then it loads it and does the neccessary recovery operations.

Author

Matija Večenaj

Parameters

none	

Returns

no value

5.55.2.3 AK_recover_archive_log()

```
void AK_recover_archive_log (
            char * fileName )
```

Function that reads the binary file in which last commands were saved, and executes them.

Function opens the recovery binary file and executes all commands that were saved inside the redo_log structure

Author

Dražen Bandić, update by Tomislav Turek

Parameters

fileName	- name of the archive log

Returns

no value

5.55.2.4 AK_recover_operation()

```
void AK_recover_operation (
            int sig )
```

Function that recovers and executes failed commands.

Function is called when SIGINT signal is sent to the system. All commands that are written to rec.bin file are recovered to the designated structure and then executed.

Author

Tomislav Turek

Parameters

sig	required integer parameter for SIGINT handler functions

5.55.2.5 AK_recovery_insert_row()

```
void AK_recovery_insert_row (
            char * table,
            int commandNumber )
```

Function that inserts a new row in the table with attributes.

Function is given the table name with desired data that should be inserted inside. By using the table name, function retrieves table attributes names and their types which uses afterwards for insert_data_test function to insert data to designated table.

Author

Dražen Bandić, updated by Tomislav Turek

Parameters

table	- table name to insert to
commandNumber	- number of current command

Returns

no value

5.55.2.6 AK_recovery_test()

```
TestResult 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

5.55.2.7 AK_recovery_tokenize()

```
char** AK_recovery_tokenize (
            char * input,
            char * delimiter,
            int valuesOrNot )
```

Function that tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)

Author

Dražen Bandić

Parameters

input	- input to tokenize
delimiter	- delimiter
valuesOrNot	- 1 if the input are values, 0 otherwise

Returns

new double pointer structure with tokens

5.55.2.8 recovery_insert_row()

```
int recovery_insert_row (
            char * table,
            char ** attr_name,
            char ** attributes,
            int n,
            int * type )
```

Function that inserts row in table.

Author

Danko Bukovac

Returns

EXIT_SUCCESS if insert is successful, else EXIT_FAILURE

5.55.3 Variable Documentation

5.55.3.1 grandfailure

```
short grandfailure = 0
```

this variable flags if system failed

5.56 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 that adds a new element to redolog.
- void **AK_redolog_commit** ()
- int AK_add_to_redolog_select (int command, struct list_node *condition, char *srcTable)

  Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.
- int AK_check_redo_log_select (int command, struct list_node *condition, char *srcTable)

  Function that checks redolog for select, works only with selection.c, not select.c.
- void AK_printout_redolog ()

  Function that prints out the content of redolog memory.
- char * AK_check_attributes (char *attributes)

  Function that checks if the attribute contains '|', and if it does it replaces it with "\|".

5.56.1 Detailed Description

Provides redolog functions.

5.56.2 Function Documentation

5.56.2.1 AK_add_to_redolog()

```
int AK_add_to_redolog (
            int command,
            struct list_node * row_root )
```

Function that adds a 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

5.56.2.2 AK_add_to_redolog_select()

```
int AK_add_to_redolog_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.

Author

Danko Bukovac

Returns

EXIT_FAILURE if not allocated memory for ispis, otherwise EXIT_SUCCESS

5.56.2.3 AK_check_attributes()

```
char* AK_check_attributes (
            char * attributes )
```

Function that checks if the attribute contains '|', and if it does it replaces it with "\|".

Author

Dražen Bandić

Returns

new attribute

5.56.2.4 AK_check_redo_log_select()

```
int AK_check_redo_log_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that checks redolog for select, works only with selection.c, not select.c.

Author

Danko Bukovac

Returns

0 if select was not found, otherwise 1

5.56.2.5 AK_printout_redolog()

```
void AK_printout_redolog ( )
```

Function that prints out the content of redolog memory.

Author

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

Returns

No return value.

5.57 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)

  Function that 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 that calculates how many attributes there are in the header with a while loop.
- void AK_agg_input_init (AK_agg_input *input)

  Function that 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 that 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 that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.
- void AK_agg_input_fix (AK_agg_input *input)

  function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -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 that aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK↩ _AVG_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.
- TestResult AK_aggregation_test ()

5.57.1 Detailed Description

Provides functions for aggregation and grouping

5.57.2 Function Documentation

5.57.2.1 AK_agg_input_add()

```
int AK_agg_input_add (
              AK_header header,
              int agg_task,
              AK_agg_input * input )
```

Function that adds a header with a task in input object for aggregation.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.57.2.2 AK_agg_input_add_to_beginning()

```
int AK_agg_input_add_to_beginning (
              AK_header header,
              int agg_task,
              AK_agg_input * input )
```

Function that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.57.2.3 AK_agg_input_fix()

```
void AK_agg_input_fix (
            AK_agg_input * input )
```

function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TAS↩K_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

input	the input object

Returns

No return value

5.57.2.4 AK_agg_input_init()

```
void AK_agg_input_init (
            AK_agg_input * input )
```

Function that initializes the input object for aggregation with init values.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.57.2.5 AK_aggregation()

```
int AK_aggregation (
            AK_agg_input * input,
            char * source_table,
            char * agg_table )
```

Function that aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK↩ _AVG_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.

Author

Dejan Frankovic

Parameters

input	input object with list of atributes by which we aggregate and types of aggregations
source_table	- table name for the source table
agg_table	table name for aggregated table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK_insert_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK_insert_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

5.57.2.6 AK_aggregation_test()

```
TestResult AK_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

5.57.2.7 AK_header_size()

```
int AK_header_size (
            AK_header * header )
```

Function that calculates how many attributes there are in the header with a while loop.

Author

Dejan Frankovic

Parameters

header	A header array

Returns

Number of attributes defined in header array

5.57.2.8 AK_search_unsorted()

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

Function that 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_TI↩ ME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

Author

Miroslav Policki

Parameters

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

Returns

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

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

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

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

5.58 rel/aggregation.h File Reference

```
#include "../auxi/test.h"
#include "selection.h"
#include "projection.h"
#include "../file/filesearch.h"
#include "../auxi/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 atribute name, date and aggregation task associated.
- struct AK_agg_input

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

Macros

- #define **AGG_TASK_GROUP** 1
- #define **AGG_TASK_COUNT** 2
- #define **AGG_TASK_SUM** 3
- #define **AGG_TASK_MAX** 4
- #define **AGG_TASK_MIN** 5
- #define **AGG_TASK_AVG** 6
- #define **AGG_TASK_AVG_COUNT** 10
- #define **AGG_TASK_AVG_SUM** 11

Functions

- int AK_header_size (AK_header *)

  Function that calculates how many attributes there are in the header with a while loop.
- void AK_agg_input_init (AK_agg_input *input)

  Function that 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 that 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 that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.
- void AK_agg_input_fix (AK_agg_input *input)

  function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -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 that aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK↩ _AVG_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.
- TestResult AK_aggregation_test ()

5.58.1 Detailed Description

Header file that provides data structures, functions and defines for aggregation and grouping

5.58.2 Function Documentation

5.58.2.1 AK_agg_input_add()

```
int AK_agg_input_add (
            AK_header header,
            int agg_task,
            AK_agg_input * input )
```

Function that adds a header with a task in input object for aggregation.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.58.2.2 AK_agg_input_add_to_beginning()

```
int AK_agg_input_add_to_beginning (
            AK_header header,
            int agg_task,
            AK_agg_input * input )
```

Function that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.

Author

Dejan Frankovic

Parameters

header	a header that is being aggregated
agg_task	the task which is to be done on the header
input	the input object

Returns

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

5.58.2.3 AK_agg_input_fix()

```
void AK_agg_input_fix (
            AK_agg_input * input )
```

function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TAS↩K_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

input	the input object

Returns

No return value

5.58.2.4 AK_agg_input_init()

```
void AK_agg_input_init (
            AK_agg_input * input )
```

Function that initializes the input object for aggregation with init values.

Author

Dejan Frankovic

Parameters

input	the input object

Returns

No return value

5.58.2.5 AK_aggregation()

```
int AK_aggregation (
            AK_agg_input * input,
            char * source_table,
            char * agg_table )
```

Function that aggregates a given table by given attributes. Firstly, AGG_TASK_AVG_COUNT and AGG_TASK↩ _AVG_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assignes the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed_values array and results are put in new table.

Author

Dejan Frankovic

Parameters

input	input object with list of atributes by which we aggregate and types of aggregations
source_table	- table name for the source table
agg_table	table name for aggregated table

Returns

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK_insert_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK_insert_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

5.58.2.6 AK_aggregation_test()

```
TestResult AK_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

5.58.2.7 AK_header_size()

```
int AK_header_size (
            AK_header * header )
```

Function that calculates how many attributes there are in the header with a while loop.

Author

Dejan Frankovic

Parameters

header	A header array

Returns

Number of attributes defined in header array

5.59 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 that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.
- TestResult AK_op_difference_test ()

  Function for difference operator testing.

5.59.1 Detailed Description

Provides functions for relational difference operation

5.59.2 Function Documentation

5.59.2.1 AK_difference()

```
int AK_difference (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.59.2.2 AK_op_difference_test()

```
TestResult AK_op_difference_test ( )
```

Function for difference operator testing.

Author

Dino Laktašić

5.60 rel/difference.h File Reference

```
#include "../auxi/test.h"
#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 that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.
- TestResult AK_op_difference_test ()

  Function for difference operator testing.

5.60.1 Detailed Description

Header file that provides functions and defines for relational difference operation

5.60.2 Function Documentation

5.60.2.1 AK_difference()

```
int AK_difference (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.60.2.2 AK_op_difference_test()

```
TestResult AK_op_difference_test ( )
```

Function for difference operator testing.

Author

Dino Laktašić

5.61 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 that 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 charachter 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 a given string input.
- int AK_check_regex_operator_expression (const char ∗value, const char ∗expression)

  Function that evaluates regex expression on a given string input.
- int AK_check_if_row_satisfies_expression (struct list_node ∗row_root, struct list_node ∗expr)

  Function that 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.
- TestResult **AK_expression_check_test** ()

5.61.1 Detailed Description

Provides functions for constraint checking used in selection and theta-join

5.61.2 Function Documentation

5.61.2.1 AK_check_arithmetic_statement()

```
int AK_check_arithmetic_statement (
            struct list_node * el,
            const char * op,
            const char * a,
            const char * b )
```

Function that 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.

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

Author

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

5.61.2.2 AK_check_if_row_satisfies_expression()

```
int AK_check_if_row_satisfies_expression (
            struct list_node * row_root,
            struct list_node * expr )
```

Function that 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.

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

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

Parameters

row_root	beginning of the row that is to be evaluated
∗expr	list with the logical expression in postfix notation

Returns

0 if row does not satisfy, 1 if row satisfies expression

5.61.2.3 AK_check_regex_expression()

```
int AK_check_regex_expression (
            const char * value,
            const char * expression,
            int sensitive,
            int checkWildCard )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression
checkWildCard	replaces SQL wildcard to correesponding POSIX regex charachter
sensitive	case insensitive indicator 1-case sensitive,0- case insensitive

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.61.2.4 AK_check_regex_operator_expression()

```
int AK_check_regex_operator_expression (
            const char * value,
            const char * expression )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.61.2.5 AK_replace_wild_card()

```
char* AK_replace_wild_card (
            const char * s,
            char ch,
            const char * repl )
```

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

@Author Leon Palaić

Parameters

s	input string
ch	charachter to be replaced

Returns

new sequence of charachters

5.62 rel/expression_check.h File Reference

```
#include "../auxi/test.h"
```

```
#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 that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.
- int AK_check_if_row_satisfies_expression (struct list_node ∗row_root, struct list_node ∗expr)

  Function that replaces charachter 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 a given string input.
- int AK_check_regex_operator_expression (const char ∗value, const char ∗expression)

  Function that evaluates regex expression on a given string input.
- TestResult **AK_expression_check_test** ()

5.62.1 Detailed Description

Header file that functions and defines for expression ckecking

5.62.2 Function Documentation

5.62.2.1 AK_check_arithmetic_statement()

```
int AK_check_arithmetic_statement (
            struct list_node * el,
            const char * op,
            const char * a,
            const char * b )
```

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

Author

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

Author

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

5.62.2.2 AK_check_if_row_satisfies_expression()

```
int AK_check_if_row_satisfies_expression (
            struct list_node * row_root,
            struct list_node * expr )
```

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

@Author Leon Palaić

Parameters

s	input string
ch	charachter to be replaced

Returns

new sequence of charachters

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

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

Parameters

row_root	beginning of the row that is to be evaluated
∗expr	list with the logical expression in postfix notation

Returns

0 if row does not satisfy, 1 if row satisfies expression

5.62.2.3 AK_check_regex_expression()

```
int AK_check_regex_expression (
            const char * value,
            const char * expression,
            int sensitive,
            int checkWildCard )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression
checkWildCard	replaces SQL wildcard to correesponding POSIX regex charachter
sensitive	case insensitive indicator 1-case sensitive,0- case insensitive

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.62.2.4 AK_check_regex_operator_expression()

```
int AK_check_regex_operator_expression (
            const char * value,
            const char * expression )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

Parameters

value	string value that must match regex expression
expression	POSIX regex expression

Returns

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

5.63 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 that makes a 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)
- TestResult AK_op_intersect_test ()

  Function for intersect operator testing.

5.63.1 Detailed Description

Provides functions for relational intersect operation

5.63.2 Function Documentation

5.63.2.1 AK_intersect()

```
int AK_intersect (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.63.2.2 AK_op_intersect_test()

```
TestResult AK_op_intersect_test ( )
```

Function for intersect operator testing.

Author

Dino Laktašić

Returns

No return value

5.64 rel/intersect.h File Reference

```
#include "../auxi/test.h"
#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 that makes a 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)
- TestResult AK_op_intersect_test ()

  Function for intersect operator testing.

5.64.1 Detailed Description

Provides data structures, functions and defines for relational intersect operation

5.64.2 Function Documentation

5.64.2.1 AK_intersect()

```
int AK_intersect (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.64.2.2 AK_op_intersect_test()

```
TestResult AK_op_intersect_test ( )
```

Function for intersect operator testing.

Author

Dino Laktašić

Returns

No return value

5.65 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 that makes a 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.
- void AK_copy_blocks_join (AK_block *tbl1_temp_block, AK_block *tbl2_temp_block, struct list_node *att, char *new_table)

  Function that 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 that makes a nat_join betwen two tables on some attributes.
- TestResult AK_op_join_test ()

  Function for natural join testing.

5.65.1 Detailed Description

Provides functions for relational natural join operation

5.65.2 Function Documentation

5.65.2.1 AK_copy_blocks_join()

```
void AK_copy_blocks_join (
            AK_block * tbl1_temp_block,
            AK_block * tbl2_temp_block,
            struct list_node * att,
            char * new_table )
```

Function that iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
att	attributes on which we make nat_join
new_table	name of the nat_join table

Returns

No return value

5.65.2.2 AK_create_join_block_header()

```
void AK_create_join_block_header (
             int table_address1,
             int table_address2,
             char * new_table,
             struct list_node * att )
```

Function that makes a header for the new table and call the function to create the segment.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

table_address1	address of the block of the first table
table_address2	address of the block of the second table
new_table	name of the join table
att_root	ttributes on which we make nat_join

Returns

No return value

5.65.2.3 AK_join()

```
int AK_join (
             char * srcTable1,
             char * srcTable2,
             char * dstTable,
             struct list_node * att )
```

Function that makes a nat_join betwen two tables on some attributes.

Author

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

Parameters

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

Returns

if success returns EXIT_SUCCESS

5.65.2.4 AK_merge_block_join()

```
void AK_merge_block_join (
            struct list_node * row_root,
            struct list_node * row_root_insert,
            AK_block * temp_block,
            char * new_table )
```

Function that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.

Author

Matija Novak, updated by Dino Laktašić

Parameters

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

Returns

No return value

5.65.2.5 AK_op_join_test()

```
TestResult AK_op_join_test ( )
```

Function for natural join testing.

Author

Matija Novak

Returns

No return value

5.66 rel/nat_join.h File Reference

```
#include "../auxi/test.h"
#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 that makes a 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.
- void AK_copy_blocks_join (AK_block ∗tbl1_temp_block, AK_block ∗tbl2_temp_block, struct list_node ∗att, char ∗new_table)

  Function that 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 that makes a nat_join betwen two tables on some attributes.
- TestResult AK_op_join_test ()

  Function for natural join testing.

5.66.1 Detailed Description

Header file that provides functions and defines for relational natural join operation

5.66.2 Function Documentation

5.66.2.1 AK_copy_blocks_join()

```
void AK_copy_blocks_join (
            AK_block * tbl1_temp_block,
            AK_block * tbl2_temp_block,
            struct list_node * att,
            char * new_table )
```

Function that iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

tbl1_temp_block	block of the first table
tbl2_temp_block	block of the second join table
att	attributes on which we make nat_join
new_table	name of the nat_join table

Returns

No return value

5.66.2.2 AK_create_join_block_header()

```
void AK_create_join_block_header (
            int table_address1,
            int table_address2,
            char * new_table,
            struct list_node * att )
```

Function that makes a header for the new table and call the function to create the segment.

Author

Matija Novak, optimized, and updated to work with AK_list by Dino Laktašić

Parameters

table_address1	address of the block of the first table
table_address2	address of the block of the second table
new_table	name of the join table
att_root	ttributes on which we make nat_join

Returns

No return value

5.66.2.3 AK_join()

```
int AK_join (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            struct list_node * att )
```

Function that makes a nat_join betwen two tables on some attributes.

Author

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

Parameters

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

Returns

if success returns EXIT_SUCCESS

5.66.2.4 AK_merge_block_join()

```
void AK_merge_block_join (
            struct list_node * row_root,
            struct list_node * row_root_insert,
            AK_block * temp_block,
            char * new_table )
```

Function that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.

Author

Matija Novak, updated by Dino Laktašić

Parameters

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

Returns

No return value

5.66.2.5 AK_op_join_test()

```
TestResult AK_op_join_test ( )
```

Function for natural join testing.

Author

Matija Novak

Returns

No return value

5.67 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 that makes the structure of an empty destination table for product operation.
- void AK_product_procedure (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, AK_header header[MAX_ATTRIBUTES])

  Functions that iterates trough both tables and concates rows. The result is in destination table.
- TestResult AK_op_product_test ()

  Function for product operator testing.

5.67.1 Detailed Description

Provides functions for relational product operation

5.67.2 Function Documentation

5.67.2.1 AK_op_product_test()

```
TestResult AK_op_product_test ( )
```

Function for product operator testing.

Author

Dino Laktašić, Fabijan Josip Kraljić

5.67.2.2 AK_product()

```
int AK_product (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes the structure of an empty destination table for product operation.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.67.2.3 AK_product_procedure()

```
void AK_product_procedure (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            AK_header header[MAX_ATTRIBUTES] )
```

Functions that iterates trough both tables and concates rows. The result is in destination table.

Author

Dino Laktašić, Fabijan Josip Kraljić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table
header	header of product table

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

5.68 rel/product.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/files.h"
#include "../auxi/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 that makes the structure of an empty destination table for product operation.

- void AK_product_procedure (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, AK_header header[MAX_ATTRIBUTES])

  Functions that iterates trough both tables and concates rows. The result is in destination table.

- TestResult AK_op_product_test ()

  Function for product operator testing.

5.68.1 Detailed Description

Header file that provides functions and defines for relational product operation

5.68.2 Function Documentation

5.68.2.1 AK_op_product_test()

```
TestResult AK_op_product_test ( )
```

Function for product operator testing.

Author

Dino Laktašić, Fabijan Josip Kraljić

5.68.2.2 AK_product()

```
int AK_product (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes the structure of an empty destination table for product operation.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.68.2.3 AK_product_procedure()

```
void AK_product_procedure (
             char * srcTable1,
             char * srcTable2,
             char * dstTable,
             AK_header header[MAX_ATTRIBUTES] )
```

Functions that iterates trough both tables and concates rows. The result is in destination table.

Author

Dino Laktašić, Fabijan Josip Kraljić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the product table
header	header of product table

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

5.69 rel/projection.c File Reference

```
#include "projection.h"
```

Include dependency graph for projection.c:

Functions

- void AK_create_block_header (int old_block, char *dstTable, struct list_node *att)

  Function that creates a new header for the projection table.
- char * AK_get_operator (char *exp)

  Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.
- void AK_remove_substring (char *s, const char *substring)

  Function that iterates through given string and removes specified part of that string.
- int AK_determine_header_type (int firstOperand, int secondOperand)

  Function that determines the new header type.
- char * AK_create_header_name (char *first, char *second, char *operator)

  Function that 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 that copies the data from old table block to the new projection table.
- char * AK_perform_operation (char *op, struct AK_operand *firstOperand, struct AK_operand *secondOperand, int type)

  Function that performes arithmetics operation depended on given operator.
- int AK_projection (char *srcTable, char *dstTable, struct list_node *att, struct list_node *expr)

  Function that makes a projection of some table on given attributes.
- TestResult AK_op_projection_test ()

  Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

5.69.1 Detailed Description

Provides functions for relational projection operation

5.69.2 Function Documentation

5.69.2.1 AK_copy_block_projection()

```
void AK_copy_block_projection (
            AK_block * old_block,
            struct list_node * att,
            char * dstTable,
            struct list_node * expr )
```

Function that copies the data from old table block to the new projection table.

Author

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

Parameters

old_block	block from which we copy data
dstTable	name of the new table
att	list of the attributes which should the projection table contain
expr	given expression to check

Returns

New projection table that contains all blocks from old table

No return value

5.69.2.2 AK_create_block_header()

```
void AK_create_block_header (
             int old_block,
             char * dstTable,
             struct list_node * att )
```

Function that creates a new header for the projection table.

Author

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

Parameters

old_block_add	address of the block from which we copy headers we need
dstTable	name of the new table - destination table
att	list of the attributes which should the projection table contain

Returns

Newly created header

No return value

5.69.2.3 AK_create_header_name()

```
char* AK_create_header_name (
             char * first,
             char * second,
             char * operator )
```

Function that creates new header name from passed operand names and operator.

Author

Leon Palaić

Parameters

first	operand name
second	operand name
operator	given operator

Returns

Function returns set of characters that represent new header name

Character - new name

5.69.2.4 AK_determine_header_type()

```
int AK_determine_header_type (
            int firstOperand,
            int secondOperand )
```

Function that determines the new header type.

Author

Leon Palaić

Parameters

firstOperand	operand type
secondOperand	operand type

Returns

Function returns determinated header type

Integer - type

5.69.2.5 AK_get_operator()

```
char* AK_get_operator (
            char * exp )
```

Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.

Author

Leon Palaić

Parameters

exp	input expression string

Returns

character - aritmetic operator

character

5.69.2.6 AK_op_projection_test()

```
TestResult AK_op_projection_test ( )
```

Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

Author

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

Returns

Projection tables and number od passed tests

Test result - number of successful and unsuccessful tests

5.69.2.7 AK_perform_operation()

```
char* AK_perform_operation (
            char * op,
            struct AK_operand * firstOperand,
            struct AK_operand * secondOperand,
            int type )
```

Function that performes arithmetics operation depended on given operator.

Author

Leon Palaić

Parameters

firstOperand	first operand
secondOperand	second operand
op	aritmetic operator
type	type of operand

Returns

result of arithmetic operation

character

5.69.2.8 AK_projection()

```
int AK_projection (
            char * srcTable,
```

```
            char * dstTable,
            struct list_node * att,
            struct list_node * expr )
```

Function that makes a projection of some table on given attributes.

Author

Matija Novak, rewritten and optimized by Dino Laktašić, now support cacheing

Parameters

srcTable	source table - table on which projection is made
expr	given expression to check while doing projection
att	list of atributes on which we make projection
dstTable	table name for projection table - new table - destination table

Returns

Projection table on given attributes

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

5.69.2.9 AK_remove_substring()

```
void AK_remove_substring (
            char * s,
            const char * substring )
```

Function that iterates through given string and removes specified part of that string.

Author

Leon Palaić

Parameters

s	input string
substring	string that needs to be removed

Returns

Cleaned new string

No return value

5.70 rel/projection.h File Reference

```
#include "../auxi/test.h"
```

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

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

Classes

- struct AK_operand

Functions

- void AK_create_block_header (int old_block, char *dstTable, struct list_node *att)

  Function that creates a new header for the projection table.
- char * AK_get_operator (char *exp)

  Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.
- void AK_remove_substring (char *s, const char *substring)

  Function that iterates through given string and removes specified part of that string.
- int AK_determine_header_type (int firstOperand, int secondOperand)

  Function that determines the new header type.
- char * AK_create_header_name (char *first, char *operator, char *second)

  Function that 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 that copies the data from old table block to the new projection table.
- char * AK_perform_operation (char *op, struct AK_operand *firstOperand, struct AK_operand *secondOperand, int type)

  Function that performes arithmetics operation depended on given operator.
- int AK_projection (char *srcTable, char *dstTable, struct list_node *att, struct list_node *expr)

  Function that makes a projection of some table on given attributes.
- TestResult AK_op_projection_test ()

  Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

5.70.1 Detailed Description

Header file that provides data structures, functions and defines for relational projection operation

5.70.2 Function Documentation

5.70.2.1 AK_copy_block_projection()

```
void AK_copy_block_projection (
            AK_block * old_block,
            struct list_node * att,
            char * dstTable,
            struct list_node * expr )
```

Function that copies the data from old table block to the new projection table.

Author

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

Parameters

old_block	block from which we copy data
dstTable	name of the new table
att	list of the attributes which should the projection table contain
expr	given expression to check

Returns

New projection table that contains all blocks from old table

No return value

5.70.2.2 AK_create_block_header()

```
void AK_create_block_header (
            int old_block,
            char * dstTable,
            struct list_node * att )
```

Function that creates a new header for the projection table.

Author

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

Parameters

old_block_add	address of the block from which we copy headers we need
dstTable	name of the new table - destination table
att	list of the attributes which should the projection table contain

Returns

Newly created header

No return value

5.70.2.3 AK_create_header_name()

```
char* AK_create_header_name (
            char * first,
            char * second,
            char * operator )
```

Function that creates new header name from passed operand names and operator.

Author

Leon Palaić

Parameters

first	operand name
second	operand name
operator	given operator

Returns

Function returns set of characters that represent new header name

Character - new name

5.70.2.4 AK_determine_header_type()

```
int AK_determine_header_type (
            int firstOperand,
            int secondOperand )
```

Function that determines the new header type.

Author

Leon Palaić

Parameters

firstOperand	operand type
secondOperand	operand type

Returns

Function returns determinated header type

Integer - type

5.70.2.5 AK_get_operator()

```
char* AK_get_operator (
            char * exp )
```

Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.

Author

Leon Palaić

Parameters

exp	input expression string

Returns

character - aritmetic operator

character

Author

Leon Palaić

Parameters

exp	input expression string

Returns

character - aritmetic operator

character

5.70.2.6 AK_op_projection_test()

```
TestResult AK_op_projection_test ( )
```

Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

Author

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

Returns

Projection tables and number od passed tests

Test result - number of successful and unsuccessful tests

5.70.2.7 AK_perform_operation()

```
char* AK_perform_operation (
            char * op,
            struct AK_operand * firstOperand,
            struct AK_operand * secondOperand,
            int type )
```

Function that performes arithmetics operation depended on given operator.

Author

Leon Palaić

Parameters

firstOperand	first operand
secondOperand	second operand
op	aritmetic operator
type	type of operand

Returns

result of arithmetic operation

character

5.70.2.8 AK_projection()

```
int AK_projection (
            char * srcTable,
            char * dstTable,
            struct list_node * att,
            struct list_node * expr )
```

Function that makes a projection of some table on given attributes.

Author

Matija Novak, rewritten and optimized by Dino Laktašić, now support cacheing

Parameters

srcTable	source table - table on which projection is made
expr	given expression to check while doing projection
att	list of atributes on which we make projection
dstTable	table name for projection table - new table - destination table

Returns

Projection table on given attributes

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

5.70.2.9 AK_remove_substring()

```
void AK_remove_substring (
            char * s,
            const char * substring )
```

Function that iterates through given string and removes specified part of that string.

Author

Leon Palaić

Parameters

s	input string
substring	string that needs to be removed

Returns

Cleaned new string

No return value

5.71 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 that which implements selection.
- TestResult AK_op_selection_test ()

Function for selection operator testing using WHERE clause and operators BETWEEN, AND.

- TestResult AK_op_selection_test_pattern ()

  Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

- int AK_selection_op_rename (char *srcTable, char *dstTable, struct list_node *expr)

  Function that which implements selection rename operation test.

5.71.1 Detailed Description

Provides functions for relational selection operation

5.71.2 Function Documentation

5.71.2.1 AK_op_selection_test()

```
TestResult AK_op_selection_test ( )
```

Function for selection operator testing using WHERE clause and operators BETWEEN, AND.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

5.71.2.2 AK_op_selection_test_pattern()

```
TestResult AK_op_selection_test_pattern ( )
```

Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

Author

Krunoslav Bilić updated by Filip Belinić

5.71.2.3 AK_selection()

```
int AK_selection (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection.

Author

Matija Šestak.

Parameters

∗srcTable	source table name
∗dstTable	destination table name
∗expr	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.71.2.4 AK_selection_op_rename()

```
int AK_selection_op_rename (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection rename operation test.

Author

unknown

Parameters

∗srcTable	source table name
∗dstTable	destination table name
∗expr	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.72 rel/selection.h File Reference

```
#include "../auxi/test.h"
#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 that which implements selection.
- TestResult AK_op_selection_test ()

  Function for selection operator testing using WHERE clause and operators BETWEEN, AND.
- TestResult AK_op_selection_test_pattern ()

  Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

5.72.1 Detailed Description

Header file that provides functions and defines for relational selection operation

5.72.2 Function Documentation

5.72.2.1 AK_op_selection_test()

```
TestResult AK_op_selection_test ( )
```

Function for selection operator testing using WHERE clause and operators BETWEEN, AND.

Author

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

5.72.2.2 AK_op_selection_test_pattern()

```
TestResult AK_op_selection_test_pattern ( )
```

Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

Author

Krunoslav Bilić updated by Filip Belinić

5.72.2.3 AK_selection()

```
int AK_selection (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection.

Author

Matija Šestak.

Parameters

∗srcTable	source table name
∗dstTable	destination table name
∗expr	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.73 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 that creates a 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 that 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 that creates a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.
- TestResult AK_op_theta_join_test ()

  Function for testing the theta join.

5.73.1 Detailed Description

Provides functions for relational theta join operation

5.73.2 Function Documentation

5.73.2.1 AK_check_constraints()

```
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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

Author

Tomislav Mikulček

Parameters

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

Returns

No return value

5.73.2.2 AK_create_theta_join_header()

```
int AK_create_theta_join_header (
            char * srcTable1,
            char * srcTable2,
            char * new_table )
```

Function that creates a header of the new table for theta join.

Author

Tomislav Mikulček

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
new_table	name of the destination table

Returns

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

5.73.2.3 AK_op_theta_join_test()

```
TestResult AK_op_theta_join_test ( )
```

Function for testing the theta join.

Author

Tomislav Mikulček

Returns

No return value

5.73.2.4 AK_theta_join()

```
int AK_theta_join (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            struct list_node * constraints )
```

Function that creates a theta join betwen two tables on specified conditions. Names of the attibutes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints
must come from the two source tables and not from a third.

Function that creates a theta join betwen two tables on specified conditions.

Author

Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

5.74 rel/theta_join.h File Reference

```
#include "../auxi/test.h"
#include "expression_check.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

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

Functions

- int AK_theta_join (char *srcTable1, char *srcTable2, char *dstTable, struct list_node *constraints)

  Function that creates a theta join betwen two tables on specified conditions.
- int AK_create_theta_join_header (char *srcTable1, char *srcTable2, char *new_table)

  Function that creates a 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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.
- TestResult AK_op_theta_join_test ()

  Function for testing the theta join.

5.74.1 Detailed Description

Header file that provides functions and defines for theta-join

5.74.2 Function Documentation

5.74.2.1 AK_check_constraints()

```
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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

Author

Tomislav Mikulček

Parameters

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

Returns

No return value

5.74.2.2 AK_create_theta_join_header()

```
int AK_create_theta_join_header (
            char * srcTable1,
            char * srcTable2,
            char * new_table )
```

Function that creates a header of the new table for theta join.

Author

Tomislav Mikulček

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
new_table	name of the destination table

Returns

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

5.74.2.3 AK_op_theta_join_test()

```
TestResult AK_op_theta_join_test ( )
```

Function for testing the theta join.

Author

Tomislav Mikulček

Returns

No return value

5.74.2.4 AK_theta_join()

```
int AK_theta_join (
              char * srcTable1,
              char * srcTable2,
              char * dstTable,
              struct list_node * constraints )
```

Function that creates a theta join betwen two tables on specified conditions.

Author

Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

Function that creates a theta join betwen two tables on specified conditions.

Author

Tomislav Mikulček,updated by Nikola Miljancic

Parameters

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

Returns

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

5.75 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 that makes a union of two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)
- TestResult AK_op_union_test ()

  Function for union operator testing.

5.75.1 Detailed Description

Provides functions for relational union operation

5.75.2 Function Documentation

5.75.2.1 AK_op_union_test()

```
TestResult AK_op_union_test ( )
```

Function for union operator testing.

Author

Dino Laktašić

Returns

No return value

5.75.2.2 AK_union()

```
int AK_union (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a union of two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

Function that makes a union of two tables.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.76 rel/union.h File Reference

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

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

Functions

- int AK_union (char *srcTable1, char *srcTable2, char *dstTable)

  Function that makes a union of two tables.
- TestResult AK_op_union_test ()

  Function for union operator testing.

5.76.1 Detailed Description

Header file that provides functions and defines for relational union operation

5.76.2 Function Documentation

5.76.2.1 AK_op_union_test()

```
TestResult AK_op_union_test ( )
```

Function for union operator testing.

Author

Dino Laktašić

Returns

No return value

5.76.2.2 AK_union()

```
int AK_union (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a union of two tables.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

Function that makes a union of two tables.

Author

Dino Laktašić

Parameters

srcTable1	name of the first table
srcTable2	name of the second table
dstTable	name of the new table

Returns

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

5.77 sql/command.c File Reference

```
#include "command.h"
```

Include dependency graph for command.c:

Functions

- int AK_command (command *commands, int commandNum)

  Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)
- TestResult AK_test_command ()

  Function for testing commands.

5.77.1 Detailed Description

TODO: Description

5.77.2 Function Documentation

5.77.2.1 AK_command()

```
int AK_command (
            command * commands,
            int commandNum )
```

Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)

Author

Mario Kolmacic updated by Ivan Pusic and Tomislav Ilisevic

Parameters

commands	Commands array to execute
commandNum	Number of commands in array

Returns

ERROR_EXIT only if command can't be executed returns EXIT_ERROR

5.77.2.2 AK_test_command()

```
TestResult AK_test_command ( )
```

Function for testing commands.

Author

Unknown, updated by Tomislav Ilisevic

5.78 sql/command.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/selection.h"
#include "../auxi/mempro.h"
```

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

Classes

- struct AK_command_struct

Typedefs

- typedef struct AK_command_struct **command**

Functions

- int AK_command (command *komande, int brojkomandi)

  Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)
- TestResult AK_test_command ()

  Function for testing commands.

5.78.1 Detailed Description

Header file that provides data structures, functions and defines for command.c

5.78.2 Function Documentation

5.78.2.1 AK_command()

```
int AK_command (
            command * commands,
            int commandNum )
```

Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)

Author

Mario Kolmacic updated by Ivan Pusic and Tomislav Ilisevic

Parameters

commands	Commands array to execute
commandNum	Number of commands in array

Returns

ERROR_EXIT only if command can't be executed returns EXIT_ERROR

5.78.2.2 AK_test_command()

```
TestResult AK_test_command ( )
```

Function for testing commands.

Author

Unknown, updated by Tomislav Ilisevic

5.79 sql/cs/between.c File Reference

```
#include "between.h"
```

Include dependency graph for between.c:

Functions

- int AK_find_table_address (char *_systemTableName)

  Function that returns system tables addresses by name.
- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  Function that sets between constraints on particular attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.
- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  Function that 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.
- TestResult AK_constraint_between_test ()

  Function that tests the functionality of implemented between constraint.

5.79.1 Detailed Description

Provides functions for between constaint

5.79.2 Function Documentation

5.79.2.1 AK_constraint_between_test()

```
TestResult AK_constraint_between_test ( )
```

Function that tests the functionality of implemented between constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Returns

No return value

5.79.2.2 AK_delete_constraint_between()

```
int AK_delete_constraint_between (
            char * tableName,
            char * constraintNamePar,
            char * attNamePar )
```

Function for deleting specific between constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.79.2.3 AK_find_table_address()

```
int AK_find_table_address (
            char * _systemTableName )
```

Function that returns system tables addresses by name.

Author

Mislav Jurinić

Parameters

_systemTableName	table name

Returns

int

5.79.2.4 AK_print_constraints()

```
void AK_print_constraints (
             char * tableName )
```

Function for printing tables.

Author

Maja Vračan

Parameters

tableName	name of table

5.79.2.5 AK_read_constraint_between()

```
int AK_read_constraint_between (
             char * tableName,
             char * newValue,
             char * attNamePar )
```

Function that checks if the given value is between lower and upper bounds of the "between" constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
newValue	value we want to insert
attNamePar	attribute name

Returns

EXIT_SUCCESS or EXIT_ERROR

5.79.2.6 AK_set_constraint_between()

```
void AK_set_constraint_between (
            char * tableName,
            char * constraintName,
            char * attName,
            char * startValue,
            char * endValue )
```

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase. It searches for AK_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
constraintName	name of constraint
attName	name of attribute
startValue	initial constraint
endValue	final constraint

Returns

No return value

5.80 sql/cs/between.h File Reference

```
#include "../../auxi/test.h"
#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)

Function that returns system tables addresses by name.

- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  Function that 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[])

  Function for deleting specific between constraint.

- TestResult AK_constraint_between_test ()

  Function that tests the functionality of implemented between constraint.

5.80.1 Detailed Description

Header file that provides functions and defines for between constaint

5.80.2 Function Documentation

5.80.2.1 AK_constraint_between_test()

```
TestResult AK_constraint_between_test ( )
```

Function that tests the functionality of implemented between constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Returns

No return value

5.80.2.2 AK_delete_constraint_between()

```
int AK_delete_constraint_between (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function for deleting specific between constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.80.2.3 AK_find_table_address()

```
int AK_find_table_address (
            char * _systemTableName )
```

Function that returns system tables addresses by name.

Author

Mislav Jurinić

Parameters

_systemTableName	table name

Returns

int

5.80.2.4 AK_read_constraint_between()

```
int AK_read_constraint_between (
            char * tableName,
            char * newValue,
            char * attNamePar )
```

Function that checks if the given value is between lower and upper bounds of the "between" constraint.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
newValue	value we want to insert
attNamePar	attribute name

Returns

EXIT_SUCCESS or EXIT_ERROR

5.80.2.5 AK_set_constraint_between()

```
void AK_set_constraint_between (
            char * tableName,
            char * constraintName,
            char * attName,
            char * startValue,
            char * endValue )
```

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
constraintName	name of constraint
attName	name of attribute
startValue	initial constraint
endValue	final constraint

Returns

No return value

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

Author

Saša Vukšić, updated by Mislav Jurinić

Parameters

tableName	table name
constraintName	name of constraint
attName	name of attribute
startValue	initial constraint
endValue	final constraint

Returns

No return value

5.81 sql/cs/check_constraint.c File Reference

```
#include "check_constraint.h"
#include "../drop.h"
```

Include dependency graph for check_constraint.c:

Functions

- int condition_passed (char *condition, int type, void *value, void *row_data)

  Function that for a 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)

  Function that adds a new "check" constraint into the system table.
- int AK_check_constraint (char *table, char *attribute, void *value)

  Function that verifies if the value we want to insert satisfies the "check" constraint.
- TestResult AK_check_constraint_test ()

  Test function for "check" constraint.

5.81.1 Detailed Description

Check constraint implementation file.

5.81.2 Function Documentation

5.81.2.1 AK_check_constraint()

```
int AK_check_constraint (
            char * table,
            char * attribute,
            void * value )
```

Function that verifies if the value we want to insert satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

table	target table name
attribute	target attribute name
value	data we want to insert

Returns

1 - result, 0 - failure

5.81.2.2 AK_check_constraint_test()

```
TestResult AK_check_constraint_test ( )
```

Test function for "check" constraint.

Author

Mislav Jurinić

Returns

void

5.81.2.3 AK_set_check_constraint()

```
int AK_set_check_constraint (
            char * table_name,
            char * constraint_name,
            char * attribute_name,
            char * condition,
            int type,
            void * value )
```

Function that adds a new "check" constraint into the system table.

Author

Mislav Jurinić

Parameters

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

Returns

1 - result, 0 - failure

5.81.2.4 condition_passed()

```
int condition_passed (
            char * condition,
            int type,
            void * value,
            void * row_data )
```

Function that for a given value, checks if it satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

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

Returns

1 - result, 0 - failure

5.82 sql/cs/check_constraint.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../rel/expression_check.h"
#include "../../auxi/mempro.h"
```

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

Functions

- int condition_passed (char ∗condition, int type, void ∗value, void ∗row_data)

  Function that for a 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)

  Function that adds a new "check" constraint into the system table.
- int AK_check_constraint (char ∗table, char ∗attribute, void ∗value)

  Function that verifies if the value we want to insert satisfies the "check" constraint.
- TestResult AK_check_constraint_test ()

  Test function for "check" constraint.

5.82.1 Detailed Description

Header file that provides functions and defines for check constraint

5.82.2 Function Documentation

5.82.2.1 AK_check_constraint()

```
int AK_check_constraint (
            char * table,
            char * attribute,
            void * value )
```

Function that verifies if the value we want to insert satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

table	target table name
attribute	target attribute name
value	data we want to insert

Returns

1 - result, 0 - failure

5.82.2.2 AK_check_constraint_test()

```
TestResult AK_check_constraint_test ( )
```

Test function for "check" constraint.

Author

Mislav Jurinić

Returns

void

5.82.2.3 AK_set_check_constraint()

```
int AK_set_check_constraint (
            char * table_name,
            char * constraint_name,
            char * attribute_name,
            char * condition,
            int type,
            void * value )
```

Function that adds a new "check" constraint into the system table.

Author

Mislav Jurinić

Parameters

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

Returns

1 - result, 0 - failure

5.82.2.4 condition_passed()

```
int condition_passed (
            char * condition,
            int type,
            void * value,
            void * row_data )
```

Function that for a given value, checks if it satisfies the "check" constraint.

Author

Mislav Jurinić

Parameters

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

Returns

1 - result, 0 - failure

5.83 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 that checks if constraint name would be unique in database.
- TestResult AK_constraint_names_test ()

  Function that tests if constraint name would be unique in database.

5.83.1 Detailed Description

Provides functions for checking if constraint name is unique in database

5.83.2 Function Documentation

5.83.2.1 AK_check_constraint_name()

```
int AK_check_constraint_name (
            char * constraintName )
```

Function that checks if constraint name would be unique in database.

Author

Nenad Makar, updated by Matej Lipovača

Parameters

char	constraintName name which you want to give to constraint which you are trying to create

Returns

EXIT_ERROR or EXIT_SUCCESS

Updated by Matej Lipovača Added other constraint names from catalog, aswell in "constants.h"

5.83.2.2 AK_constraint_names_test()

```
TestResult AK_constraint_names_test ( )
```

Function that tests if constraint name would be unique in database.

Author

Nenad Makar

Returns

No return value

5.84 sql/cs/constraint_names.h File Reference

```
#include "../../auxi/test.h"
#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 that checks if constraint name would be unique in database.
- TestResult AK_constraint_names_test ()
  Function that tests if constraint name would be unique in database.

5.84.1 Detailed Description

Header file that provides functions and defines for checking if constraint name is unique in database

5.84.2 Function Documentation

5.84.2.1 AK_check_constraint_name()

```
int AK_check_constraint_name (
            char * constraintName )
```

Function that checks if constraint name would be unique in database.

Author

Nenad Makar, updated by Mislav Jurinić

Parameters

char	constraintName name which you want to give to constraint which you are trying to create

Returns

EXIT_ERROR or EXIT_SUCCESS

Author

Nenad Makar, updated by Matej Lipovača

Parameters

char	constraintName name which you want to give to constraint which you are trying to create

Returns

EXIT_ERROR or EXIT_SUCCESS

Updated by Matej Lipovača Added other constraint names from catalog, aswell in "constants.h"

5.84.2.2 AK_constraint_names_test()

```
TestResult AK_constraint_names_test ( )
```

Function that tests if constraint name would be unique in database.

Author

Nenad Makar

Returns

No return value

5.85 sql/cs/nnull.c File Reference

```
#include "nnull.h"
```

Include dependency graph for nnull.c:

Functions

- int AK_set_constraint_not_null (char *tableName, char *attName, char *constraintName)

  Function that sets NOT NULL constraint on an attribute.
- int AK_check_constraint_not_null (char *tableName, char *attName, char *constraintName)

  Function that checks if constraint name is unique and in violation of NOT NULL constraint.
- int AK_read_constraint_not_null (char *tableName, char *attName, char *newValue)

  Function checks if NOT NULL constraint is already set.
- int AK_delete_constraint_not_null (char *tableName, char attName[ ], char constraintName[ ])

  Function for deleting specific not null constraint.
- TestResult AK_nnull_constraint_test ()

  Function for testing NOT NULL constraint.

5.85.1 Detailed Description

Provides functions for not null constraint

5.85.2 Function Documentation

5.85.2.1 AK_check_constraint_not_null()

```
int AK_check_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that checks if constraint name is unique and in violation of NOT NULL constraint.

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

5.85.2.2 AK_delete_constraint_not_null()

```
int AK_delete_constraint_not_null (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function for deleting specific not null constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.85.2.3 AK_nnull_constraint_test()

```
TestResult AK_nnull_constraint_test ( )
```

Function for testing NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Returns

No return value

5.85.2.4 AK_read_constraint_not_null()

```
int AK_read_constraint_not_null (
            char * tableName,
            char * attName,
            char * newValue )
```

Function checks if NOT NULL constraint is already set.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char∗	attName name of attribute
char∗	newValue new value

Returns

EXIT_ERROR or EXIT_SUCCESS

5.85.2.5 AK_set_constraint_not_null()

```
int AK_set_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that sets NOT NULL constraint on an 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

5.86 sql/cs/nnull.h File Reference

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

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

Functions

- int AK_set_constraint_not_null (char ∗tableName, char ∗attName, char ∗constraintName)

Function that sets NOT NULL constraint on an attribute.

- int AK_read_constraint_not_null (char *tableName, char *attName, char *newValue)

  Function checks if NOT NULL constraint is already set.
- int AK_check_constraint_not_null (char *tableName, char *attName, char *newValue)

  Function that checks if constraint name is unique and in violation of NOT NULL constraint.
- int AK_delete_constraint_not_null (char *tableName, char attName[], char constraintName[])

  Function for deleting specific not null constraint.
- TestResult AK_nnull_constraint_test ()

  Function for testing NOT NULL constraint.

5.86.1 Detailed Description

Header file that provides functions and defines for not null constraint

5.86.2 Function Documentation

5.86.2.1 AK_check_constraint_not_null()

```
int AK_check_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that checks if constraint name is unique and in violation of NOT NULL constraint.

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

5.86.2.2 AK_delete_constraint_not_null()

```
int AK_delete_constraint_not_null (
            char * tableName,
```

```
            char attName[],
            char constraintName[] )
```

Function for deleting specific not null constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.86.2.3 AK_nnull_constraint_test()

```
TestResult AK_nnull_constraint_test ( )
```

Function for testing NOT NULL constraint.

Author

Saša Vukšić, updated by Nenad Makar

Returns

No return value

5.86.2.4 AK_read_constraint_not_null()

```
int AK_read_constraint_not_null (
            char * tableName,
            char * attName,
            char * newValue )
```

Function checks if NOT NULL constraint is already set.

Author

Saša Vukšić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char∗	attName name of attribute
char∗	newValue new value

Returns

EXIT_ERROR or EXIT_SUCCESS

5.86.2.5 AK_set_constraint_not_null()

```
int AK_set_constraint_not_null (
             char * tableName,
             char * attName,
             char * constraintName )
```

Function that sets NOT NULL constraint on an 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

5.87 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 that 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 that reads a reference entry from system table.

- int AK_reference_check_attribute (char *tableName, char *attribute, char *value)

  Function that checks referential integrity for one attribute.

- int AK_reference_check_if_update_needed (struct list_node *lista, int action)

  Funcction 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 that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

- int AK_reference_update (struct list_node *lista, int action)

  Function that updates child table entries according to ongoing update of parent table entries.

- int AK_reference_check_entry (struct list_node *lista)

  Function that checks a new entry for referential integrity.

- TestResult AK_reference_test ()

  Function for testing referential integrity.

5.87.1 Detailed Description

Provides functions for referential integrity

5.87.2 Function Documentation

5.87.2.1 AK_add_reference()

```
int AK_add_reference (
            char * childTable,
            char * childAttNames[],
            char * parentTable,
            char * parentAttNames[],
            int attNum,
            char * constraintName,
            int type )
```

Function that adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name.

Author

Dejan Frankovic

Parameters

name	of the child table
array	of child table attribute names (foreign key attributes)
name	of the parent table
array	of parent table attribute names (primary key attributes)
number	of attributes in foreign key
name	of the constraint
type	of the constraint, constants defined in 'reference.h'

Returns

EXIT_SUCCESS

5.87.2.2 AK_get_reference()

```
AK_ref_item AK_get_reference (
             char * tableName,
             char * constraintName )
```

Function that reads a reference entry from system table.

Author

Dejan Frankovic

Parameters

name	of the table with reference (with foreign key)
name	of the reference constraint

Returns

AK_ref_item object with all neccessary information about the reference

5.87.2.3 AK_reference_check_attribute()

```
int AK_reference_check_attribute (
             char * tableName,
             char * attribute,
             char * value )
```

Function that checks referential integrity for one attribute.

Author

Dejan Frankovic

Parameters

child	table name
attribute	name (foreign key attribute)
value	of the attribute we're checking

Returns

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

5.87.2.4 AK_reference_check_entry()

```
int AK_reference_check_entry (
            struct list_node * lista )
```

Function that checks a new entry for referential integrity.

Author

Dejan Franković

Parameters

list	of elements for insert row

Returns

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

5.87.2.5 AK_reference_check_if_update_needed()

```
int AK_reference_check_if_update_needed (
            struct list_node * lista,
            int action )
```

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

Author

Dejan Frankovic

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

5.87.2.6 AK_reference_check_restricion()

```
int AK_reference_check_restricion (
            struct list_node * lista,
            int action )
```

Function that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE?

Returns

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

5.87.2.7 AK_reference_test()

```
TestResult AK_reference_test ( )
```

Function for testing referential integrity.

Author

Dejan Franković

Returns

No return value

5.87.2.8 AK_reference_update()

```
int AK_reference_update (
            struct list_node * lista,
            int action )
```

Function that updates child table entries according to ongoing update of parent table entries.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS

5.88 sql/cs/reference.h File Reference

```
#include "../../auxi/test.h"
#include "../../dm/dbman.h"
#include "../../file/table.h"
#include "../../auxi/mempro.h"
```

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

Classes

- struct AK_ref_item

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

Macros

- #define REF_TYPE_NONE -1

  Constant declaring none reference type.
- #define REF_TYPE_SET_NULL 1

  Constant declaring set null reference type.
- #define REF_TYPE_NO_ACTION 2

  Constant declaring no action reference type.
- #define **REF_TYPE_CASCADE** 3
- #define REF_TYPE_RESTRICT 4

  Constant declaring restrict reference type.
- #define REF_TYPE_SET_DEFAULT 5

  Constant declaring set default reference type.
- #define MAX_REFERENCE_ATTRIBUTES 10

  Constant declaring maximum number of reference attributes.
- #define MAX_CHILD_CONSTRAINTS 20

  Constant declaring maximum number of child constraints.

Functions

- int AK_add_reference (char *childTable, char *childAttNames[ ], char *parentTable, char *parentAttNames[ ], int attNum, char *constraintName, int type)

  Function that 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 that reads a reference entry from system table.
- int AK_reference_check_attribute (char *tableName, char *attribute, char *value)

  Function that checks referential integrity for one attribute.
- int AK_reference_check_if_update_needed (struct list_node *lista, int action)

  Funcction 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 that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.
- int AK_reference_update (struct list_node *lista, int action)

  Function that updates child table entries according to ongoing update of parent table entries.
- int AK_reference_check_entry (struct list_node *lista)

  Function that checks a new entry for referential integrity.
- TestResult 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)

  Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.
- void AK_Update_Existing_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  Used to add a constraint attribute which will define what element gets updated when the operation is executed.
- 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 that 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)

  !! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.
- int AK_delete_row (struct list_node *row_root)

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

  Function updates rows of some table.
- int AK_initialize_new_segment (char *name, int type, AK_header *header)

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

5.88.1 Detailed Description

đ Provides data structures, functions and defines for referential integrity

5.88.2 Macro Definition Documentation

5.88.2.1 REF_TYPE_NO_ACTION

```
#define REF_TYPE_NO_ACTION 2
```

Constant declaring no action reference type.

Constant declaring cascade reference type.

5.88.3 Function Documentation

5.88.3.1 AK_add_reference()

```
int AK_add_reference (
             char * childTable,
             char * childAttNames[],
             char * parentTable,
             char * parentAttNames[],
             int attNum,
             char * constraintName,
             int type )
```

Function that adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name.

Author

Dejan Frankovic

Parameters

name	of the child table
array	of child table attribute names (foreign key attributes)
name	of the parent table
array	of parent table attribute names (primary key attributes)
number	of attributes in foreign key
name	of the constraint
type	of the constraint, constants defined in 'reference.h'

Returns

EXIT_SUCCESS

5.88.3.2 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row @returs EXIT_SUCCESS if success

5.88.3.3 AK_get_reference()

```
AK_ref_item AK_get_reference (
            char * tableName,
            char * constraintName )
```

Function that reads a reference entry from system table.

Author

Dejan Frankovic

Parameters

name	of the table with reference (with foreign key)
name	of the reference constraint

Returns

AK_ref_item object with all neccessary information about the reference

5.88.3.4 AK_initialize_new_segment()

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

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

Author

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

Parameters

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

Returns

start address of new segment

5.88.3.5 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.

Author

Matija Novak, changed by Dino Laktašić

Parameters

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

Returns

No return value

5.88.3.6 AK_Insert_New_Element_For_Update()

```
void AK_Insert_New_Element_For_Update (
            int newtype,
```

```
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore,
            int newconstraint )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

Author

Matija Novak

Parameters

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

Returns

No return value

5.88.3.7 AK_insert_row()

```
int AK_insert_row (
            struct list_node * row_root )
```

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

Author

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

Parameters

row_root	list of elements which contain data of one row

Returns

EXIT_SUCCESS if success else EXIT_ERROR

5.88.3.8 AK_reference_check_attribute()

```
int AK_reference_check_attribute (
            char * tableName,
            char * attribute,
            char * value )
```

Function that checks referential integrity for one attribute.

Author

Dejan Frankovic

Parameters

child	table name
attribute	name (foreign key attribute)
value	of the attribute we're checking

Returns

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

5.88.3.9 AK_reference_check_entry()

```
int AK_reference_check_entry (
            struct list_node * lista )
```

Function that checks a new entry for referential integrity.

Author

Dejan Franković

Parameters

list	of elements for insert row

Returns

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

5.88.3.10 AK_reference_check_if_update_needed()

```
int AK_reference_check_if_update_needed (
            struct list_node * lista,
            int action )
```

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

Author

Dejan Frankovic

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

5.88.3.11 AK_reference_check_restricion()

```
int AK_reference_check_restricion (
            struct list_node * lista,
            int action )
```

Function that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE?

Returns

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

5.88.3.12 AK_reference_test()

```
TestResult AK_reference_test ( )
```

Function for testing referential integrity.

Author

Dejan Franković

Returns

No return value

5.88.3.13 AK_reference_update()

```
int AK_reference_update (
            struct list_node * lista,
            int action )
```

Function that updates child table entries according to ongoing update of parent table entries.

Author

Dejan Franković

Parameters

list	of elements for update
is	action UPDATE or DELETE ?

Returns

EXIT_SUCCESS

5.88.3.14 AK_selection()

```
int AK_selection (
            char * srcTable,
```

```
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection.

Author

Matija Šestak.

Parameters

∗srcTable	source table name
∗dstTable	destination table name
∗expr	list with posfix notation of the logical expression

Returns

EXIT_SUCCESS

5.88.3.15 AK_Update_Existing_Element()

```
void AK_Update_Existing_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a constraint attribute which will define what element gets updated when the operation is executed.

Author

Igor Rinkovec

Parameters

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

Returns

No return value

5.88.3.16 AK_update_row()

```
int AK_update_row (
            struct list_node * row_root )
```

Function updates rows of some table.

Author

Matija Novak, Dejan Frankovic (added referential integrity)

Parameters

row_root	elements of one row

Returns

EXIT_SUCCESS if success

5.89 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 that sets unique constraint on attribute(s)
- int AK_read_constraint_unique (char *tableName, char attName[], char newValue[])

  Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.
- int AK_delete_constraint_unique (char *tableName, char attName[], char constraintName[])

  Function for deleting specific unique constraint.
- TestResult AK_unique_test ()

  Function for testing UNIQUE constraint.

5.89.1 Detailed Description

Provides functions for unique constraint

5.89.2 Function Documentation

5.89.2.1 AK_delete_constraint_unique()

```
int AK_delete_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function for deleting specific unique constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.89.2.2 AK_read_constraint_unique()

```
int AK_read_constraint_unique (
            char * tableName,
            char attName[],
            char newValue[] )
```

Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

char∗	tableName name of table
char	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test)
char	newValue[] new value(s), if you want to check combination of values of more attributes seperate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using AK_tuple_to_string(struct list_node ∗tuple) or with sprintf in a similiar way it's used in that function (if value isn't part of a ∗tuple), to concatenate more values in newValue[] use strcat(destination, source) and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

Returns

EXIT_ERROR or EXIT_SUCCESS

5.89.2.3 AK_set_constraint_unique()

```
int AK_set_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function that sets unique constraint on attribute(s)

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

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

Returns

EXIT_ERROR or EXIT_SUCCESS

5.89.2.4 AK_unique_test()

```
TestResult AK_unique_test ( )
```

Function for testing UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Returns

No return value

5.90 sql/cs/unique.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
#include "../../auxi/dictionary.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 that sets unique constraint on attribute(s)
- int AK_read_constraint_unique (char *tableName, char attName[ ], char newValue[ ])

  Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.
- int AK_delete_constraint_unique (char *tableName, char attName[ ], char constraintName[ ])

  Function for deleting specific unique constraint.
- TestResult AK_unique_test ()

  Function for testing UNIQUE constraint.

5.90.1 Detailed Description

Header file that provides functions and defines for unique constraint

5.90.2 Function Documentation

5.90.2.1 AK_delete_constraint_unique()

```
int AK_delete_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function for deleting specific unique constraint.

Author

Maja Vračan

Parameters

tableName	name of table on which constraint refers
attName	name of attribute on which constraint is declared
constraintName	name of constraint

Returns

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

5.90.2.2 AK_read_constraint_unique()

```
int AK_read_constraint_unique (
            char * tableName,
            char attName[],
            char newValue[] )
```

Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

*char**	tableName name of table
char	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test)
char	newValue[] new value(s)

Returns

EXIT_ERROR or EXIT_SUCCESS

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

*char**	tableName name of table
char	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test)
char	newValue[] new value(s), if you want to check combination of values of more attributes seperate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using AK_tuple_to_string(struct list_node *tuple) or with sprintf in a similiar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use strcat(destination, source) and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

Returns

EXIT_ERROR or EXIT_SUCCESS

5.90.2.3 AK_set_constraint_unique()

```
int AK_set_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function that sets unique constraint on attribute(s)

Author

Domagoj Tuličić, updated by Nenad Makar

Parameters

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

Returns

EXIT_ERROR or EXIT_SUCCESS

5.90.2.4 AK_unique_test()

```
TestResult AK_unique_test ( )
```

Function for testing UNIQUE constraint.

Author

Domagoj Tuličić, updated by Nenad Makar

Returns

No return value

5.91 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 the 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 checking if the element(view, function, sequence, user ...) exist in system catalog table.
- TestResult AK_drop_test ()

  Function for testing all DROP functions.

Variables

- char * **system_catalog** [NUM_SYS_TABLES]

5.91.1 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

5.91.2 Function Documentation

5.91.2.1 AK_drop()

```
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

type	drop type
drop_arguments	arguments of DROP command

5.91.2.2 AK_drop_help_function()

```
void AK_drop_help_function (
            char * tblName,
            char * sys_table )
```

Help function for the drop command. Delete memory blocks and addresses of table and removes table or index from system table.

Author

unknown, Jurica Hlevnjak - fix bugs and reorganize code in this function

Parameters

tblName	name of table or index
sys_table	name of system catalog table

5.91.2.3 AK_drop_test()

```
TestResult 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

5.91.2.4 AK_if_exist()

```
int AK_if_exist (
            char * tblName,
            char * sys_table )
```

Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.

Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

tblName	name of table, index view, function, trigger, sequence, user, group or constraint
sys_table	name of system catalog table

Returns

if element exist in system catalog returns 1, if not returns 0

5.91.3 Variable Documentation

5.91.3.1 system_catalog

```
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"
}
```

5.92 sql/drop.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/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.
- TestResult AK_drop_test ()

  Function for testing all DROP functions.
- int AK_if_exist (char *tblName, char *sys_table)

  Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.

5.92.1 Detailed Description

Header file that provides data structures, functions and defines for unique constraint

5.92.2 Function Documentation

5.92.2.1 AK_drop()

```
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

type	drop type
drop_arguments	arguments of DROP command

5.92.2.2 AK_drop_test()

```
TestResult 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

5.92.2.3 AK_if_exist()

```
int AK_if_exist (
            char * tblName,
            char * sys_table )
```

Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.

Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

Parameters

tblName	name of table, index view, function, trigger, sequence, user, group or constraint
sys_table	name of system catalog table

Returns

if element exist in system catalog returns 1, if not returns 0

5.93 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 belongs to a function.
- int AK_check_function_arguments_type (int function_id, struct list_node *args)

  Function that checks whether arguments belongs to a 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 that removes a function by its obj_id.
- int AK_function_arguments_remove_by_obj_id (int *obj_id)

  Function that 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.
- TestResult AK_function_test ()

  Function for functions testing.

5.93.1 Detailed Description

Provides functions for functions

5.93.2 Function Documentation

5.93.2.1 AK_check_function_arguments()

```
int AK_check_function_arguments (
            int function_id,
            struct list_node * arguments_list )
```

Function that checks whether arguments belongs to a function.

Author

Boris Kišić

Parameters

*function_id	id of the function
*arguments_list	list of arguments

Returns

EXIT_SUCCESS of the function or EXIT_ERROR

5.93.2.2 AK_check_function_arguments_type()

```
int AK_check_function_arguments_type (
            int function_id,
            struct list_node * args )
```

Function that checks whether arguments belongs to a function but only checks argument type (not name). Used for drop function.

Author

Jurica Hlevnjak updated by Aleksandra Polak

Parameters

function_id	id of the function
args	function arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.3 AK_function_add()

```
int AK_function_add (
            char * name,
            int return_type,
            struct list_node * arguments_list )
```

Function that adds a function to system table.

Author

Boris Kišić, updated by Tomislav Ilisevic

Parameters

*name	name of the function
*return_type	data type returned from a function - values from 0 to 13 - defined in constants.h
*arguments_list	list of function arguments

Returns

function id or EXIT_ERROR

5.93.2.4 AK_function_arguments_add()

```
int AK_function_arguments_add (
            int function_id,
            int arg_number,
            int arg_type,
            char * argname )
```

Function that adds a function argument to system table.

Author

Boris Kišić

Parameters

*function_id	id of the function to which the argument belongs
*arg_number	number of the argument
*arg_type	data type of the argument
*argname	name of the argument

Returns

function argument id or EXIT_ERROR

5.93.2.5 AK_function_arguments_remove_by_obj_id()

```
int AK_function_arguments_remove_by_obj_id (
            int * obj_id )
```

Function that removes function arguments by function id.

Author

Boris Kišić

Parameters

obj↩ _id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.6 AK_function_change_return_type()

```
int AK_function_change_return_type (
            char * name,
            struct list_node * arguments_list,
            int new_return_type )
```

Function that changes the return type.

Author

Boris Kišić

Parameters

∗name	name of the function to be modified
∗arguments_list	list of function arguments
∗new_return_type	new return type

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.7 AK_function_remove_by_name()

```
int AK_function_remove_by_name (
            char * name,
            struct list_node * arguments_list )
```

Function that removes a function from system table by name and arguments.

Author

Boris Kišić

Parameters

∗name	name of the function
∗arguments_list	list of arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.8 AK_function_remove_by_obj_id()

```
int AK_function_remove_by_obj_id (
            int obj_id )
```

Function that removes a function by its obj_id.

Author

Boris Kišić

Parameters

obj_id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.9 AK_function_rename()

```
int AK_function_rename (
            char * name,
            struct list_node * arguments_list,
            char * new_name )
```

Function that changes the function name.

Author

Boris Kišić

Parameters

∗name	name of the function to be modified
∗arguments_list	list of arguments to be modified
∗new_name	new name of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.93.2.10 AK_function_test()

```
TestResult AK_function_test ( )
```

Function for functions testing.

Author

Boris Kišić, updated by Tomislav Ilisevic

Returns

No return value

5.93.2.11 AK_get_function_obj_id()

```
int AK_get_function_obj_id (
            char * function,
            struct list_node * arguments_list )
```

Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).

Author

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic

Parameters

∗function	name of the function
∗arguments_list	list of arguments

Returns

obj_id of the function or EXIT_ERROR

5.94 sql/function.h File Reference

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

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

Functions

- int AK_get_function_obj_id (char *function, struct list_node *arguments_list)

  Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).
- int AK_check_function_arguments (int function_id, struct list_node *arguments_list)

  Function that checks whether arguments belongs to a function.
- int AK_check_function_arguments_type (int function_id, struct list_node *args)

  Function that checks whether arguments belongs to a 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 that removes a function by its obj_id.
- int AK_function_arguments_remove_by_obj_id (int *obj_id)

  Function that 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.
- TestResult AK_function_test ()

  Function for functions testing.

5.94.1 Detailed Description

Header file that provides functions and defines for functions

Header file that provides functions and defines for view.c

5.94.2 Function Documentation

5.94.2.1 AK_check_function_arguments()

```
int AK_check_function_arguments (
            int function_id,
            struct list_node * arguments_list )
```

Function that checks whether arguments belongs to a function.

Author

Boris Kišić

Parameters

∗*function_id*	id of the function
∗*arguments_list*	list of arguments

Returns

EXIT_SUCCESS of the function or EXIT_ERROR

5.94.2.2 AK_check_function_arguments_type()

```
int AK_check_function_arguments_type (
            int function_id,
            struct list_node * args )
```

Function that checks whether arguments belongs to a function but only checks argument type (not name). Used for drop function.

Author

Jurica Hlevnjak updated by Aleksandra Polak

Parameters

function↩ _id	id of the function
args	function arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.3 AK_function_add()

```
int AK_function_add (
            char * name,
            int return_type,
            struct list_node * arguments_list )
```

Function that adds a function to system table.

Author

Boris Kišić, updated by Tomislav Ilisevic

Parameters

∗*name*	name of the function
∗*return_type*	data type returned from a function - values from 0 to 13 - defined in constants.h
∗*arguments_list*	list of function arguments

Returns

function id or EXIT_ERROR

5.94.2.4 AK_function_arguments_add()

```
int AK_function_arguments_add (
              int function_id,
              int arg_number,
              int arg_type,
              char * argname )
```

Function that adds a function argument to system table.

Author

Boris Kišić

Parameters

∗*function_id*	id of the function to which the argument belongs
∗*arg_number*	number of the argument
∗*arg_type*	data type of the argument
∗*argname*	name of the argument

Returns

function argument id or EXIT_ERROR

5.94.2.5 AK_function_arguments_remove_by_obj_id()

```
int AK_function_arguments_remove_by_obj_id (
              int * obj_id )
```

Function that removes function arguments by function id.

Author

Boris Kišić

Parameters

obj↩ _id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.6 AK_function_change_return_type()

```
int AK_function_change_return_type (
            char * name,
            struct list_node * arguments_list,
            int new_return_type )
```

Function that changes the return type.

Author

Boris Kišić

Parameters

∗name	name of the function to be modified
∗arguments_list	list of function arguments
∗new_return_type	new return type

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.7 AK_function_remove_by_name()

```
int AK_function_remove_by_name (
            char * name,
            struct list_node * arguments_list )
```

Function that removes a function from system table by name and arguments.

Author

Boris Kišić

Parameters

∗*name*	name of the function
∗*arguments_list*	list of arguments

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.8 AK_function_remove_by_obj_id()

```
int AK_function_remove_by_obj_id (
            int obj_id )
```

Function that removes a function by its obj_id.

Author

Boris Kišić

Parameters

obj↩ _id	obj_id of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.9 AK_function_rename()

```
int AK_function_rename (
            char * name,
            struct list_node * arguments_list,
            char * new_name )
```

Function that changes the function name.

Author

Boris Kišić

Parameters

∗*name*	name of the function to be modified
∗*arguments_list*	list of arguments to be modified
∗*new_name*	new name of the function

Returns

EXIT_SUCCESS or EXIT_ERROR

5.94.2.10 AK_function_test()

```
TestResult AK_function_test ( )
```

Function for functions testing.

Author

Boris Kišić, updated by Tomislav Ilisevic

Returns

No return value

5.94.2.11 AK_get_function_obj_id()

```
int AK_get_function_obj_id (
            char * function,
            struct list_node * arguments_list )
```

Function that gets obj_id of a function by name and arguments list (transferred from trigger.c/drop.c).

Author

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic

Parameters

∗*function*	name of the function
∗*arguments_list*	list of arguments

Returns

obj_id of the function or EXIT_ERROR

5.95 sql/insert.h File Reference

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

```
#include "../file/fileio.h"
#include "../auxi/constants.h"
#include "../file/table.h"
#include "drop.h"
```

Include dependency graph for insert.h:

Functions

- AK_header * AK_get_insert_header (int *size, char *tblName, struct list_node *columns)

  Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.
- int AK_insert (char *tableName, struct list_node *columns, struct list_node *values)

  Function that implements SQL insert command.
- TestResult **AK_insert_test** ()

5.95.1 Detailed Description

Implementation of SQL insert command.

Header file SQL insert command.

5.95.2 Function Documentation

5.95.2.1 AK_get_insert_header()

```
AK_header* AK_get_insert_header (
            int * size,
            char * tblName,
            struct list_node * columns )
```

Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.

Author

Filip Žmuk

Parameters

size	pointer to integer in which size of header will be saved
tblName	table in which rows will be inserted
columns	list of columns in SQL command

Returns

header for values to be inserted or EXIT_ERROR

5.95.2.2 AK_insert()

```
int AK_insert (
            char * tblName,
            struct list_node * columns,
            struct list_node * values )
```

Function that implements SQL insert command.

Author

Filip Žmuk

Parameters

tableName	table in which rows will be inserted
columns	list of columns
values	values to be inserted

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96 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)

  Inserts a new user in the AK_user table.
- int AK_user_get_id (char ∗username)

  Function that returns an ID of the given user.
- int AK_user_check_pass (char ∗username, int ∗password)

  Function that checks if there is user with given password.
- int AK_user_remove_by_name (char ∗name)

  Function that removes the given user.
- int AK_user_rename (char ∗old_name, char ∗new_name, int ∗password)

  Function that renames a given user.
- int AK_group_add (char ∗name, int set_id)

  Function that adds a new group.

- int AK_group_get_id (char *name)
  Function that returns the ID from the given group name.
- int AK_group_remove_by_name (char *name)
  Function that removes the given group.
- int AK_group_rename (char *old_name, char *new_name)
  Function that renames the given group.
- int AK_grant_privilege_user (char *username, char *table, char *right)
  Function that grants a specific privilege to the desired user on a given table.
- int AK_revoke_privilege_user (char *username, char *table, char *right)
  Function that revokes users privilege on the given table.
- int AK_revoke_all_privileges_user (char *username)
  Function that revokes ALL user's privileges on ALL tables (for DROP user)
- int AK_grant_privilege_group (char *groupname, char *table, char *right)
  Function that grants a privilege to a given group on a given table.
- int AK_revoke_privilege_group (char *groupname, char *table, char *right)
  Function that revokes a groups privilege on the given table.
- int AK_revoke_all_privileges_group (char *groupname)
  Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)
- int AK_add_user_to_group (char *user, char *group)
  Function that puts the desired user in the given group.
- int AK_remove_user_from_all_groups (char *user)
  Function that removes user from all groups. Used for DROP user.
- int AK_remove_all_users_from_group (char *group)
  Function that removes all users from a group. Used for DROP group.
- int AK_check_privilege (char *username, char *table, char *privilege)
  Function that checks whether the given user has a right for the given operation on the given table.
- int AK_check_user_privilege (char *user)
  Function that checks if the user has any privileges or belongs to any group. Used in drop user for restriction.
- int AK_check_group_privilege (char *group)
  Function that checks if the group has any privileges. Used in drop group for restriction.
- TestResult AK_privileges_test ()
  Function that tests all the previous functions.

5.96.1 Detailed Description

Provides functions for privileges

5.96.2 Function Documentation

5.96.2.1 AK_add_user_to_group()

```
int AK_add_user_to_group (
            char * user,
            char * group )
```

Function that puts the desired user in the given group.

Author

Kristina Takač, updated by Mario Peroković, added verifying the existence of user in the group, updated by Maja Vračan

Parameters

∗*user*	username of user which will be put in group
∗*group*	name of group in which user will be put

Returns

EXIT_SUCCESS or EXIT_ERROR if the user is already in the group

5.96.2.2 AK_check_group_privilege()

```
int AK_check_group_privilege (
            char * group )
```

Function that checks if the group has any privileges. Used in drop group for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec, updated by Marko Flajšek

Parameters

group	name of group

Returns

EXIT_ERROR or EXIT_SUCCESS

5.96.2.3 AK_check_privilege()

```
int AK_check_privilege (
            char * username,
            char * table,
            char * privilege )
```

Function that checks whether the given user has a right for the given operation on the given table.

Author

Kristina Takač, updated by Marko Flajšek

Parameters

∗*user*	username for which we want check privileges
∗*table*	name of table for which we want to check whether user has right on
∗*privilege*	privilege for which we want to check whether user has right for

Returns

EXIT_SUCCESS if user has right, EXIT_ERROR if user has no right

5.96.2.4 AK_check_user_privilege()

```
int AK_check_user_privilege (
            char * user )
```

Function that checks if the user has any privileges or belongs to any group. Used in drop user for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_ERROR or EXIT_SUCCESS

5.96.2.5 AK_grant_privilege_group()

```
int AK_grant_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that grants a privilege to a given group on a given table.

Author

Kristina Takač.

Parameters

∗*groupname*	name of group to which we want to grant privilege
∗*table*	name of table on which privilege will be granted to user
∗*right*	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.96.2.6 AK_grant_privilege_user()

```
int AK_grant_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that grants a specific privilege to the desired user on a given table.

Author

Kristina Takač, updated by Mario Peroković, inserting user id instead of username in AK_user_right, updated by Marko Flajšek

Parameters

∗username	username of user to whom we want to grant privilege
∗table	name of table on which privilege will be granted to user
∗right	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.96.2.7 AK_group_add()

```
int AK_group_add (
            char * name,
            int set_id )
```

Function that adds a new group.

Author

Kristina Takač, edited by Ljubo Barać

Parameters

∗name	name of group to be added
set_id	non default id to be passed

Returns

id of group

5.96.2.8 AK_group_get_id()

```
int AK_group_get_id (
            char * name )
```

Function that returns the ID from the given group name.

Author

Kristina Takač.

Parameters

∗name	name of group whose id we are looking for

Returns

id of group, otherwise EXIT_ERROR

5.96.2.9 AK_group_remove_by_name()

```
int AK_group_remove_by_name (
            char * name )
```

Function that removes the given group.

Author

Ljubo Barać

Parameters

name	Name of the group to be removed

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96.2.10 AK_group_rename()

```
int AK_group_rename (
            char * old_name,
            char * new_name )
```

Function that renames the given group.

Author

Ljubo Barać, update by Lidija Lastavec

Parameters

old_name	Name of the group to be renamed
new_name	New name of the group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96.2.11 AK_privileges_test()

```
TestResult AK_privileges_test ( )
```

Function that tests all the previous functions.

Author

Kristina Takač, updated by Tomislav Ilisevic, updated by Lidija Lastavec, updated by Marko Flajšek

Returns

no return value

5.96.2.12 AK_remove_all_users_from_group()

```
int AK_remove_all_users_from_group (
            char * group )
```

Function that removes all users from a group. Used for DROP group.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

group	name of group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96.2.13 AK_remove_user_from_all_groups()

```
int AK_remove_user_from_all_groups (
            char * user )
```

Function that removes user from all groups. Used for DROP user.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96.2.14 AK_revoke_all_privileges_group()

```
int AK_revoke_all_privileges_group (
            char * groupname )
```

Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)

Author

Jurica Hlevnjak

Parameters

groupname	name of group from which we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.96.2.15 AK_revoke_all_privileges_user()

```
int AK_revoke_all_privileges_user (
            char * username )
```

Function that revokes ALL user's privileges on ALL tables (for DROP user)

Author

Jurica Hlevnjak, updated by Marko Flajšek

Parameters

username	name of user from whom we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.96.2.16 AK_revoke_privilege_group()

```
int AK_revoke_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that revokes a groups privilege on the given table.

NOTICE: Test 9 isn't currently revoking a privilege since the obj_id in the AK_group_right table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id

Parameters

∗*grounamep*	name of group which user belongs to
∗*table*	name of table on which privilege will be granted to group
∗*right*	type of privilege which will be granted to group on a given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.96.2.17 AK_revoke_privilege_user()

```
int AK_revoke_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that revokes users privilege on the given table.

NOTICE: Test 12 isn't currently revoking a privilege since the obj_id in the AK_group_right table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user_id in AK_user_right

Parameters

∗*username*	username of user to whom we want to grant privilege
∗*table*	name of table on which privilege will be revoked from user
∗*right*	type of privilege which will be revoked from user on given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.96.2.18 AK_user_add()

```
int AK_user_add (
            char * username,
            int * password,
            int set_id )
```

Inserts a new user in the AK_user table.

Author

Kristina Takač.

Parameters

∗*username*	username of user to be added
∗*password*	password of user to be added
set_id	obj_id of the new user

Returns

user_id

5.96.2.19 AK_user_check_pass()

```
int AK_user_check_pass (
            char * username,
            int * password )
```

Function that checks if there is user with given password.

Author

Fran MIkolić.

Parameters

∗username	username of user whose password we are checking
∗password	password of given username whom we will check

Returns

check 0 if false or 1 if true

5.96.2.20 AK_user_get_id()

```
int AK_user_get_id (
            char * username )
```

Function that returns an ID of the given user.

Author

Kristina Takač.

Parameters

∗username	username of user whose id we are looking for

Returns

user_id, otherwise EXIT_ERROR

5.96.2.21 AK_user_remove_by_name()

```
int AK_user_remove_by_name (
            char * name )
```

Function that removes the given user.

Author

Ljubo Barać

Parameters

name	Name of the user to be removed

Returns

EXIT_SUCCESS or EXIT_ERROR

5.96.2.22 AK_user_rename()

```
int AK_user_rename (
            char * old_name,
            char * new_name,
            int * password )
```

Function that renames a given user.

Author

Ljubo Barać, update by Lidija Lastavec, update by Marko Flajšek

Parameters

old_name	Name of the user to be renamed
new_name	New name of the user
password	Password of the user to be renamed (should be provided)

Returns

EXIT_SUCCESS or EXIT_ERROR

5.97 sql/privileges.h File Reference

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

```
#include "../file/fileio.h"
#include "../file/id.h"
#include "../rec/archive_log.h"
#include "../auxi/mempro.h"
```

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

Functions

- int AK_user_add (char *username, int *password, int set_id)

  Inserts a new user in the AK_user table.
- int AK_user_get_id (char *username)

  Function that returns an ID of the given user.
- int AK_user_check_pass (char *username, int *password)

  Function that checks if there is user with given password.
- int AK_group_add (char *name, int set_id)

  Function that adds a new group.
- int AK_group_get_id (char *name)

  Function that returns the ID from the given group name.
- int AK_grant_privilege_user (char *username, char *table, char *right)

  Function that grants a specific privilege to the desired user on a given table.
- int AK_revoke_privilege_user (char *username, char *table, char *right)

  Function that revokes users privilege on the given table.
- int AK_revoke_all_privileges_user (char *username)

  Function that revokes ALL user's privileges on ALL tables (for DROP user)
- int AK_grant_privilege_group (char *groupname, char *table, char *right)

  Function that grants a privilege to a given group on a given table.
- int AK_revoke_privilege_group (char *groupname, char *table, char *right)

  Function that revokes a groups privilege on the given table.
- int AK_revoke_all_privileges_group (char *groupname)

  Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)
- int AK_add_user_to_group (char *user, char *group)

  Function that puts the desired user in the given group.
- int AK_remove_user_from_all_groups (char *user)

  Function that removes user from all groups. Used for DROP user.
- int AK_remove_all_users_from_group (char *group)

  Function that removes all users from a group. Used for DROP group.
- int AK_check_privilege (char *username, char *table, char *privilege)

  Function that checks whether the given user has a right for the given operation on the given table.
- int AK_check_user_privilege (char *user)

  Function that checks if the user has any privileges or belongs to any group. Used in drop user for restriction.
- int AK_check_group_privilege (char *group)

  Function that checks if the group has any privileges. Used in drop group for restriction.
- int AK_group_remove_by_name (char *name)

  Function that removes the given group.
- int AK_user_rename (char *old_name, char *new_name, int *password)

  Function that renames a given user.
- int AK_group_rename (char *old_name, char *new_name)

  Function that renames the given group.
- TestResult AK_privileges_test ()

  Function that tests all the previous functions.

5.97.1 Detailed Description

Header file that provides functions and defines for privileges.c

5.97.2 Function Documentation

5.97.2.1 AK_add_user_to_group()

```
int AK_add_user_to_group (
            char * user,
            char * group )
```

Function that puts the desired user in the given group.

Author

Kristina Takač, updated by Mario Peroković, added verifying the existence of user in the group, updated by Maja Vračan

Parameters

∗user	username of user which will be put in group
∗group	name of group in which user will be put

Returns

EXIT_SUCCESS or EXIT_ERROR if the user is already in the group

5.97.2.2 AK_check_group_privilege()

```
int AK_check_group_privilege (
            char * group )
```

Function that checks if the group has any privileges. Used in drop group for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec, updated by Marko Flajšek

Parameters

group	name of group

Returns

EXIT_ERROR or EXIT_SUCCESS

5.97.2.3 AK_check_privilege()

```
int AK_check_privilege (
            char * username,
            char * table,
            char * privilege )
```

Function that checks whether the given user has a right for the given operation on the given table.

Author

Kristina Takač, updated by Marko Flajšek

Parameters

∗user	username for which we want check privileges
∗table	name of table for which we want to check whether user has right on
∗privilege	privilege for which we want to check whether user has right for

Returns

EXIT_SUCCESS if user has right, EXIT_ERROR if user has no right

5.97.2.4 AK_check_user_privilege()

```
int AK_check_user_privilege (
            char * user )
```

Function that checks if the user has any privileges or belongs to any group. Used in drop user for restriction.

Author

Jurica Hlevnjak, updated by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_ERROR or EXIT_SUCCESS

5.97.2.5 AK_grant_privilege_group()

```
int AK_grant_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that grants a privilege to a given group on a given table.

Author

Kristina Takač.

Parameters

∗*groupname*	name of group to which we want to grant privilege
∗*table*	name of table on which privilege will be granted to user
∗*right*	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.97.2.6 AK_grant_privilege_user()

```
int AK_grant_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that grants a specific privilege to the desired user on a given table.

Author

Kristina Takač, updated by Mario Peroković, inserting user id instead of username in AK_user_right, updated by Marko Flajšek

Parameters

∗*username*	username of user to whom we want to grant privilege
∗*table*	name of table on which privilege will be granted to user
∗*right*	type of privilege which will be granted to user on given table

Returns

privilege_id or EXIT_ERROR if table or user aren't correct

5.97.2.7 AK_group_add()

```
int AK_group_add (
            char * name,
            int set_id )
```

Function that adds a new group.

Author

Kristina Takač, edited by Ljubo Barać

Parameters

∗name	name of group to be added
set_id	non default id to be passed

Returns

id of group

5.97.2.8 AK_group_get_id()

```
int AK_group_get_id (
            char * name )
```

Function that returns the ID from the given group name.

Author

Kristina Takač.

Parameters

∗name	name of group whose id we are looking for

Returns

id of group, otherwise EXIT_ERROR

5.97.2.9 AK_group_remove_by_name()

```
int AK_group_remove_by_name (
            char * name )
```

Function that removes the given group.

Author

Ljubo Barać

Parameters

name	Name of the group to be removed

Returns

EXIT_SUCCESS or EXIT_ERROR

5.97.2.10 AK_group_rename()

```
int AK_group_rename (
            char * old_name,
            char * new_name )
```

Function that renames the given group.

Author

Ljubo Barać, update by Lidija Lastavec

Parameters

old_name	Name of the group to be renamed
new_name	New name of the group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.97.2.11 AK_privileges_test()

```
TestResult AK_privileges_test ( )
```

Function that tests all the previous functions.

Author

Kristina Takač, updated by Tomislav Ilisevic, updated by Lidija Lastavec, updated by Marko Flajšek

Returns

no return value

5.97.2.12 AK_remove_all_users_from_group()

```
int AK_remove_all_users_from_group (
            char * group )
```

Function that removes all users from a group. Used for DROP group.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

group	name of group

Returns

EXIT_SUCCESS or EXIT_ERROR

5.97.2.13 AK_remove_user_from_all_groups()

```
int AK_remove_user_from_all_groups (
            char * user )
```

Function that removes user from all groups. Used for DROP user.

Author

Jurica Hlevnjak, update by Lidija Lastavec

Parameters

user	name of user

Returns

EXIT_SUCCESS or EXIT_ERROR

5.97.2.14 AK_revoke_all_privileges_group()

```
int AK_revoke_all_privileges_group (
            char * groupname )
```

Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)

Author

Jurica Hlevnjak

Parameters

groupname	name of group from which we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.97.2.15 AK_revoke_all_privileges_user()

```
int AK_revoke_all_privileges_user (
            char * username )
```

Function that revokes ALL user's privileges on ALL tables (for DROP user)

Author

Jurica Hlevnjak, updated by Marko Flajšek

Parameters

username	name of user from whom we want to revoke all privileges

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.97.2.16 AK_revoke_privilege_group()

```
int AK_revoke_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that revokes a groups privilege on the given table.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id

Parameters

∗grounamep	name of group which user belongs to
∗table	name of table on which privilege will be granted to group
∗right	type of privilege which will be granted to group on a given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

NOTICE: Test 9 isn't currently revoking a privilege since the obj_id in the AK_group_right table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id

Parameters

∗grounamep	name of group which user belongs to
∗table	name of table on which privilege will be granted to group
∗right	type of privilege which will be granted to group on a given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.97.2.17 AK_revoke_privilege_user()

```
int AK_revoke_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that revokes users privilege on the given table.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user_id in AK_user_right

Parameters

∗*username*	username of user to whom we want to grant privilege
∗*table*	name of table on which privilege will be revoked from user
∗*right*	type of privilege which will be revoked from user on given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

NOTICE: Test 12 isn't currently revoking a privilege since the obj_id in the AK_group_right table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

Author

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user_id in AK_user_right

Parameters

∗*username*	username of user to whom we want to grant privilege
∗*table*	name of table on which privilege will be revoked from user
∗*right*	type of privilege which will be revoked from user on given table

Returns

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

5.97.2.18 AK_user_add()

```
int AK_user_add (
            char * username,
            int * password,
            int set_id )
```

Inserts a new user in the AK_user table.

Author

Kristina Takač.

Parameters

∗*username*	username of user to be added
∗*password*	password of user to be added
set_id	obj_id of the new user

Returns

user_id

5.97.2.19 AK_user_check_pass()

```
int AK_user_check_pass (
            char * username,
            int * password )
```

Function that checks if there is user with given password.

Author

Fran MIkolić.

Parameters

∗*username*	username of user whose password we are checking
∗*password*	password of given username whom we will check

Returns

check 0 if false or 1 if true

5.97.2.20 AK_user_get_id()

```
int AK_user_get_id (
            char * username )
```

Function that returns an ID of the given user.

Author

Kristina Takač.

Parameters

∗username	username of user whose id we are looking for

Returns

user_id, otherwise EXIT_ERROR

5.97.2.21 AK_user_rename()

```
int AK_user_rename (
            char * old_name,
            char * new_name,
            int * password )
```

Function that renames a given user.

Author

Ljubo Barać, update by Lidija Lastavec, update by Marko Flajšek

Parameters

old_name	Name of the user to be renamed
new_name	New name of the user
password	Password of the user to be renamed (should be provided)

Returns

EXIT_SUCCESS or EXIT_ERROR

5.98 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, struct list_node ∗ordering)

  Function that implements SELECT relational operator.
- TestResult AK_select_test ()

  Function for testing the implementation.

5.98.1 Detailed Description

Provides functions for SELECT relational operator

5.98.2 Function Documentation

5.98.2.1 AK_select()

```
int AK_select (
            char * srcTable,
            char * destTable,
            struct list_node * attributes,
            struct list_node * condition,
            struct list_node * ordering )
```

Function that implements SELECT relational operator.

Author

Filip Žmuk

Parameters

srcTable	- original table that is used for selection
destTable	- table that contains the result
condition	- condition for selection
attributes	- atributes to be selected
ordering	- atributes for result sorting

Returns

EXIT_SUCCESS if cache result in memory and print table else break

5.98.2.2 AK_select_test()

```
TestResult AK_select_test ( )
```

Function for testing the implementation.

Author

Renata Mesaros, updatet Filip Žmuk

5.99 sql/select.h File Reference

```
#include "../file/table.h"
#include "../auxi/test.h"
#include "../file/fileio.h"
#include "../rel/selection.h"
#include "../rel/projection.h"
#include "../auxi/auxiliary.h"
#include "../auxi/mempro.h"
#include "../file/filesort.h"
```

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

Functions

- int AK_select (char *srcTable, char *destTable, struct list_node *attributes, struct list_node *condition, struct list_node *ordering)

  Function that implements SELECT relational operator.
- TestResult AK_select_test ()

  Function for testing the implementation.

5.99.1 Detailed Description

Header file that provides functions for select.h

5.99.2 Function Documentation

5.99.2.1 AK_select()

```
int AK_select (
            char * srcTable,
            char * destTable,
            struct list_node * attributes,
            struct list_node * condition,
            struct list_node * ordering )
```

Function that implements SELECT relational operator.

Author

Filip Žmuk

Parameters

srcTable	- original table that is used for selection
destTable	- table that contains the result
condition	- condition for selection
attributes	- atributes to be selected
ordering	- atributes for result sorting

Returns

EXIT_SUCCESS if cache result in memory and print table else break

5.99.2.2 AK_select_test()

```
TestResult AK_select_test ( )
```

Function for testing the implementation.

Author

Renata Mesaros, updatet Filip Žmuk

5.100 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)

  Function that saves conditions for a trigger.
- int AK_trigger_add (char *name, char *event, struct list_node *condition, char *table, char *function, struct list_node *arguments_list)

  Function that adds a trigger to the 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 the system table by name.
- int AK_trigger_remove_by_obj_id (int obj_id)

  Function that removes a trigger by its obj_id.
- int AK_trigger_edit (char *name, char *event, struct list_node *condition, char *table, char *function)

  Function that 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 that fetches postfix list of conditions for the trigger (compatible with selection)
- int AK_trigger_rename (char *old_name, char *new_name, char *table)

  Function that renames the trigger.
- TestResult AK_trigger_test ()

  Function for trigger testing.

5.100.1 Detailed Description

Provides functions for triggers

5.100.2 Function Documentation

5.100.2.1 AK_trigger_add()

```
int AK_trigger_add (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function,
            struct list_node * arguments_list )
```

Function that adds a trigger to the system table.

Author

Unknown updated by Aleksandra Polak

Parameters

∗name	name of the trigger
∗event	event that calls the trigger - this should perhaps be an integer with defined constants...
∗condition	AK_list list of conditions in postfix
∗table	name of the table trigger is hooked on
∗function	function that is being called by the trigger

Returns

trigger id or EXIT_ERROR

5.100.2.2 AK_trigger_edit()

```
int AK_trigger_edit (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function )
```

Function that 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.

Function that edits information about the trigger in system table.

Author

Unknown

Parameters

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

Returns

EXIT_SUCCESS or EXIT_ERROR @IMPORTANT: ∗AK_dbg_messg.. need to be fixed then we could uncomment lines with this function, by then we use ∗printf.

5.100.2.3 AK_trigger_get_conditions()

```
struct list_node* AK_trigger_get_conditions (
            int trigger )
```

Function that fetches postfix list of conditions for the trigger (compatible with selection)

Author

Unknown, updated by Mario Peroković

Parameters

trigger	obj_id of the trigger

Returns

list of conditions for the trigger

5.100.2.4 AK_trigger_get_id()

```
int AK_trigger_get_id (
            char * name,
            char * table )
```

Function that gets obj_id of a trigger defined by name and table.

Author

Parameters

∗name	name of the trigger
∗table	name of the table on which the trigger is hooked

Returns

obj_id of the trigger or EXIT_ERROR

5.100.2.5 AK_trigger_remove_by_name()

```
int AK_trigger_remove_by_name (
            char * name,
            char * table )
```

Function that removes a trigger from the system table by name.

Author

Unknown

Parameters

∗name	name of the trigger
∗table	name of the table

Returns

EXIT_SUCCESS or EXIT_ERROR

5.100.2.6 AK_trigger_remove_by_obj_id()

```
int AK_trigger_remove_by_obj_id (
            int obj_id )
```

Function that removes a trigger by its obj_id.

Author

Unknown

Parameters

obj_id	obj_id of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.100.2.7 AK_trigger_rename()

```
int AK_trigger_rename (
            char * old_name,
            char * new_name,
            char * table )
```

Function that renames the trigger.

Author

Ljubo Barać

Parameters

old_name	Name of the trigger to be renamed
new_name	New name of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.100.2.8 AK_trigger_save_conditions()

```
int AK_trigger_save_conditions (
            int trigger,
            struct list_node * condition )
```

Function that saves conditions for a trigger.

Author

Unknown, updated by Mario Peroković, check if data is TYPE_INT

Parameters

trigger	obj_id of the trigger in question
∗*condition*	AK_list list of conditions

Returns

EXIT_SUCCESS or EXIT_ERROR

5.100.2.9 AK_trigger_test()

```
TestResult AK_trigger_test ( )
```

Function for trigger testing.

Author

Unknown updated by Aleksandra Polak

5.101 sql/trigger.h File Reference

```
#include "../auxi/test.h"
#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)

  Function that saves conditions for a trigger.
- int AK_trigger_add (char ∗name, char ∗event, struct list_node ∗condition, char ∗table, char ∗function, struct list_node ∗arguments_list)

  Function that adds a trigger to the 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 the system table by name.
- int AK_trigger_remove_by_obj_id (int obj_id)

  Function that removes a trigger by its obj_id.
- int AK_trigger_edit (char ∗name, char ∗event, struct list_node ∗condition, char ∗table, char ∗function)

  Function that edits information about the trigger in system table.
- struct list_node ∗ AK_trigger_get_conditions (int trigger)

  Function that fetches postfix list of conditions for the trigger (compatible with selection)
- int AK_trigger_rename (char ∗old_name, char ∗new_name, char ∗table)

  Function that renames the trigger.
- TestResult AK_trigger_test ()

  Function for trigger testing.

5.101.1 Detailed Description

Header file that provides functions and defines for trigger.c

5.101.2 Function Documentation

5.101.2.1 AK_trigger_add()

```
int AK_trigger_add (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function,
            struct list_node * arguments_list )
```

Function that adds a trigger to the system table.

Author

Unknown updated by Aleksandra Polak

Parameters

∗name	name of the trigger
∗event	event that calls the trigger - this should perhaps be an integer with defined constants...
∗condition	AK_list list of conditions in postfix
∗table	name of the table trigger is hooked on
∗function	function that is being called by the trigger

Returns

trigger id or EXIT_ERROR

5.101.2.2 AK_trigger_edit()

```
int AK_trigger_edit (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function )
```

Function that edits information about the trigger in system table.

Author

Unknown

Parameters

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

Returns

EXIT_SUCCESS or EXIT_ERROR

Function that edits information about the trigger in system table.

Author

Unknown

Parameters

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

Returns

EXIT_SUCCESS or EXIT_ERROR @IMPORTANT: ∗AK_dbg_messg.. need to be fixed then we could uncomment lines with this function, by then we use ∗printf.

5.101.2.3 AK_trigger_get_conditions()

```
struct list_node* AK_trigger_get_conditions (
            int trigger )
```

Function that fetches postfix list of conditions for the trigger (compatible with selection)

Author

Unknown, updated by Mario Peroković

Parameters

trigger	obj_id of the trigger

Returns

list of conditions for the trigger

5.101.2.4 AK_trigger_get_id()

```
int AK_trigger_get_id (
            char * name,
            char * table )
```

Function that gets obj_id of a trigger defined by name and table.

Author

Parameters

∗name	name of the trigger
∗table	name of the table on which the trigger is hooked

Returns

obj_id of the trigger or EXIT_ERROR

5.101.2.5 AK_trigger_remove_by_name()

```
int AK_trigger_remove_by_name (
            char * name,
            char * table )
```

Function that removes a trigger from the system table by name.

Author

Unknown

Parameters

∗name	name of the trigger
∗table	name of the table

Returns

EXIT_SUCCESS or EXIT_ERROR

5.101.2.6 AK_trigger_remove_by_obj_id()

```
int AK_trigger_remove_by_obj_id (
            int obj_id )
```

Function that removes a trigger by its obj_id.

Author

Unknown

Parameters

obj_id	obj_id of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.101.2.7 AK_trigger_rename()

```
int AK_trigger_rename (
            char * old_name,
            char * new_name,
            char * table )
```

Function that renames the trigger.

Author

Ljubo Barać

Parameters

old_name	Name of the trigger to be renamed
new_name	New name of the trigger

Returns

EXIT_SUCCESS or EXIT_ERROR

5.101.2.8 AK_trigger_save_conditions()

```
int AK_trigger_save_conditions (
            int trigger,
            struct list_node * condition )
```

Function that saves conditions for a trigger.

Author

Unknown, updated by Mario Peroković, check if data is TYPE_INT

Parameters

trigger	obj_id of the trigger in question
∗condition	AK_list list of conditions

Returns

EXIT_SUCCESS or EXIT_ERROR

5.101.2.9 AK_trigger_test()

```
TestResult AK_trigger_test ( )
```

Function for trigger testing.

Author

Unknown updated by Aleksandra Polak

5.102 sql/view.c File Reference

```
#include "view.h"
```

Include dependency graph for view.c:

Functions

- char * AK_check_view_name (char *name)

  Function that checks if the name of the view already exists in AK_view table.
- int AK_get_view_obj_id (char *name)

  Function that finds an object's id by its name.
- char * AK_get_view_query (char *name)

  Function that returns a query by its name.
- char * AK_get_rel_exp (char *name)

  Function that returns a relation expression by its name param name name of the view.
- int AK_view_add (char *name, char *query, char *rel_exp, int set_id)

  Function that 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)

  Function that removes the view by its object id.
- int AK_view_rename (char *name, char *new_name)

  Function that renames a view (based on it's name) from "name" to "new_name".
- int AK_view_remove_by_name (char *name)

  Function that 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_change_query (char *name, char *query, char *rel_exp)

  Function that changes the query from a view (determined by it's name) to "query".
- int AK_test_get_view_data (char *rel_exp)

  Function that shows the data from test view query. Only for test purpose.
- TestResult AK_view_test ()

  A testing function for view.c functions.

5.102.1 Detailed Description

Provides functions for views

5.102.2 Function Documentation

5.102.2.1 AK_check_view_name()

```
char* AK_check_view_name (
            char * name )
```

Function that checks if the name of the view already exists in AK_view table.

Author

Sara Kisic

Parameters

name	Name of the view

Returns

EXIT_ERROR if the name already exists or name

5.102.2.2 AK_get_rel_exp()

```
char* AK_get_rel_exp (
            char * name )
```

Function that returns a relation expression by its name param name name of the view.

Author

Danko Sačer

Returns

rel_exp string or EXIT_ERROR

5.102.2.3 AK_get_view_obj_id()

```
int AK_get_view_obj_id (
            char * name )
```

Function that finds an object's id by its name.

Author

Kresimir Ivkovic

Parameters

name	name of the view

Returns

View's id or EXIT_ERROR

5.102.2.4 AK_get_view_query()

```
char* AK_get_view_query (
            char * name )
```

Function that returns a query by its name.

Author

Danko Sačer

Parameters

name	name of the view

Returns

query string or EXIT_ERROR

5.102.2.5 AK_test_get_view_data()

```
int AK_test_get_view_data (
            char * rel_exp )
```

Function that shows the data from test view query. Only for test purpose.

Author

Darko Hranic

Parameters

rel_exp	conditions as string

5.102.2.6 AK_view_add()

```
int AK_view_add (
            char * name,
            char * query,
            char * rel_exp,
            int set_id )
```

Function that adds a new view to the view table with the corresponding name and value (view query); set_id is optional, if it's not set, the system will determine the new id automatically.

Author

Kresimir Ivkovic

Parameters

name	name og the view
query	query of the view
rel_exp	relation expression of the view
set_id	id of view

Returns

Id of the newly inserted view

5.102.2.7 AK_view_change_query()

```
int AK_view_change_query (
            char * name,
            char * query,
            char * rel_exp )
```

Function that changes the query from a view (determined by it's name) to "query".

Author

Kresimir Ivkovic

Parameters

name	of the query
query	new query of the view
rel_exp	relation expression of the view

Returns

error or success

5.102.2.8 AK_view_remove_by_name()

```
int AK_view_remove_by_name (
            char * name )
```

Function that removes the view by its name by identifying the view's id and passing id to AK_view_remove_by_↩ obj_id.

Author

Kresimir Ivkovic

Parameters

name	name of the view

Returns

Result of AK_view_remove_by_obj_id or EXIT_ERROR if no id is found

5.102.2.9 AK_view_remove_by_obj_id()

```
int AK_view_remove_by_obj_id (
            int obj_id )
```

Function that removes the view by its object id.

Author

Kresimir Ivkovic

Parameters

obj_id	object id of the view

Returns

Result of AK_delete_row for the view (success or error)

5.102.2.10 AK_view_rename()

```
int AK_view_rename (
            char * name,
            char * new_name )
```

Function that renames a view (based on it's name) from "name" to "new_name".

Author

Kresimir Ivkovic

Parameters

name	name of the view
new_name	new name of the view

Returns

error or success

5.102.2.11 AK_view_test()

```
TestResult AK_view_test ( )
```

A testing function for view.c functions.

Author

Kresimir Ivkovic, updated by Lidija Lastavec

5.103 tools/comments.py File Reference

Functions

- def comments.getcommentsFiles ()

  This function is searching for file that ends with either .py extension or .c extension and appending the same in constant cFiles/pyFiles.
- def comments.detectLanguage ()

  Function is detecting language (is it croatian or alike) of a newly created commentsFile.
- def comments.makeCommentsFile ()

  Function is parsing comments from file with .c extension and .py extension.

Variables

- string **comments.commentsFile** = "all_comments.tmp"
- list **comments.cFiles** = [ ]
- list **comments.pyFiles** = [ ]

5.103.1 Detailed Description

//!

5.104 tools/getFiles.sh File Reference

5.104.1 Detailed Description

Finding all files that ends with extension .py or .c and storing them into file.txt

5.105 tools/parseC.sh File Reference

5.105.1 Detailed Description

Parsing every C file

5.106 tools/parsePy.sh File Reference

5.106.1 Detailed Description

Parsing every Py file

5.107 tools/updateVersion.sh File Reference

5.107.1 Detailed Description

Updating project version

5.108 trans/transaction.c File Reference

```
#include "transaction.h"
```

Include dependency graph for transaction.c:

Functions

- int AK_memory_block_hash (int blockMemoryAddress)

  Function that calculates the 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)

  Function that searches for a existing entry in hash list of active blocks.
- AK_transaction_elem_P AK_search_empty_link_for_hook (int blockAddress)

  Function that 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)

  Function that adds an element to the doubly linked list.
- int AK_delete_hash_entry_list (int blockAddress)

  Function that 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)

  Function that 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)

  Function that 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)

  Function that, 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)

  Function that 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)

  Function that appends all addresses affected by the transaction.

- int AK_execute_commands (command *commandArray, int lengthOfArray)

  Function 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)

  Function that is the thread start point all relevant functions. 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)

  Function that receives all the data and gives an id to that data and starts a thread that executes the transaction.

- int AK_transaction_register_observer (AK_observable_transaction *observable_transaction, AK_observer *observer)

  Function for registering new observer of AK_observable_transaction type.

- int AK_transaction_unregister_observer (AK_observable_transaction *observable_transaction, AK_observer *observer)

  Function for unregistering observer from AK_observable_transction type.

- void handle_transaction_notify (AK_observer_lock *observer_lock)

  Function for handling AK_observable_transaction notify. Function is associated to some observer instance.

- void AK_on_observable_notify (void *observer, void *observable, AK_ObservableType_Enum type)

  Function for handling notify from some observable type.

- void AK_on_transaction_end (pthread_t transaction_thread)

  Function for handling event when some transaction is finished.

- void AK_on_all_transactions_end ()

  Function for handling event when all transactions are finished.

- void AK_on_lock_release ()

  Function for handling event when one of lock is released.

- void AK_handle_observable_transaction_action (NoticeType *noticeType)

  Function for handling action which is called from observable_transaction type.

- void AK_lock_released ()

  Function which is called when the lock is released.

- void AK_transaction_finished ()

  Function that is called when some transaction is finished.

- void AK_all_transactions_finished ()

  Function that 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.

- TestResult **AK_test_Transaction** ()

Variables

- AK_transaction_list **LockTable** [NUMBER_OF_KEYS]
- pthread_mutex_t **accessLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **acquireLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **newTransactionLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **endTransationTestLockMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_cond_t **cond_lock** = PTHREAD_COND_INITIALIZER
- AK_observable_transaction * **observable_transaction**
- pthread_t **activeThreads** [MAX_ACTIVE_TRANSACTIONS_COUNT]
- int **activeTransactionsCount** = 0
- int **transactionsCount** = 0

5.108.1 Detailed Description

Defines functions for transaction execution

5.108.2 Function Documentation

5.108.2.1 AK_acquire_lock()

```
int AK_acquire_lock (
            int memoryAddress,
            int type,
            pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

Author

Frane Jakelić updated by Ivan Pusic

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

Parameters

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

Returns

OK or NOT_OK based on the success of the function.

########\n# Lock Granted after wait#\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n##########\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */

###################\n# Lock Granted #\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n##########\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */

5.108.2.2 AK_add_hash_entry_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
            int blockAddress,
            int type )
```

Function that adds an element to the doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
type	of lock issued to the provided memory address.

Returns

pointer to the newly created doubly linked element.

5.108.2.3 AK_add_lock()

```
AK_transaction_lock_elem_P AK_add_lock (
            AK_transaction_elem_P HashList,
            int type,
            pthread_t transactionId )
```

Function that adds an element to the locks doubly linked list.

Author

Frane Jakelić

Parameters

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

Returns

pointer to the newly created Locks doubly linked element.

5.108.2.4 AK_all_transactions_finished()

```
void AK_all_transactions_finished ( )
```

Function that is called when all transactions are finished.

Author

Ivan Pusic

5.108.2.5 AK_create_lock()

```
AK_transaction_lock_elem_P AK_create_lock (
            int blockAddress,
            int type,
            pthread_t transactionId )
```

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

Author

Frane Jakelić

Parameters

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

Returns

pointer to the newly created Locks doubly linked element.

5.108.2.6 AK_create_new_transaction_thread()

```
int AK_create_new_transaction_thread (
            AK_transaction_data * transaction_data )
```

Function for creating new thread. Function also adds thread ID to pthread_t array.

Author

Ivan Pusic

Parameters

transaction_data	Data for executing transaction

Returns

Exit status (OK or NOT_OK)

5.108.2.7 AK_delete_hash_entry_list()

```
int AK_delete_hash_entry_list (
            int blockAddress )
```

Function that deletes a specific element in the lockTable doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

integer OK or NOT_OK based on success of finding the specific element in the list.

5.108.2.8 AK_delete_lock_entry_list()

```
int AK_delete_lock_entry_list (
            int blockAddress,
            pthread_t id )
```

Function that deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

int OK or NOT_OK based on success of finding the specific element in the list.

5.108.2.9 AK_execute_commands()

```
int AK_execute_commands (
            command * commandArray,
            int lengthOfArray )
```

Function that is called in a separate thread that is responsible for acquiring locks, releasing them and finding the associated block addresses.

Author

Frane Jakelić updated by Ivan Pusic

Todo Check multithreading, check if it's working correctly

Parameters

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

Returns

ABORT or COMMIT based on the success of the function.

5.108.2.10 AK_execute_transaction()

```
void* AK_execute_transaction (
            void * params )
```

Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

data	transmitted to the thread from the main thread

5.108.2.11 AK_get_memory_blocks()

```
int AK_get_memory_blocks (
            char * tblName,
            AK_memoryAddresses_link addressList )
```

Function that appends all addresses affected by the transaction.

Author

Frane Jakelić

Parameters

addressList	pointer to the linked list where the addresses are stored.
tblName	table name used in the transaction

Returns

OK or NOT_OK based on the success of the function.

5.108.2.12 AK_handle_observable_transaction_action()

```
void AK_handle_observable_transaction_action (
            NoticeType * noticeType )
```

Function for handling action which is called from observable_transaction type.

Author

Ivan Pusic

Parameters

noticeType	Type of action (event)

5.108.2.13 AK_init_observable_transaction()

```
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

5.108.2.14 AK_init_observer_lock()

```
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

5.108.2.15 AK_isLock_waiting()

```
int AK_isLock_waiting (
            AK_transaction_elem_P lockHolder,
            int type,
            pthread_t transactionId,
            AK_transaction_lock_elem_P lock )
```

Function that, based on the parameters, puts an transaction action in waiting phase or let's the transaction do it's actions.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

lockHolder	pointer to the hash list entry that is entitled to the specific memory address.
type	of lock issued to the provided memory address.
transaction↩ Id	integer representation of transaction id.
lock	pointer to the lock element that is being tested.

Returns

int PASS_LOCK_QUEUE or WAIT_FOR_UNLOCK based on the rules described inside the function.

5.108.2.16 AK_lock_released()

```
void AK_lock_released ( )
```

Function which is called when the lock is released.

Author

Ivan Pusic

5.108.2.17 AK_memory_block_hash()

```
int AK_memory_block_hash (
            int blockMemoryAddress )
```

Function that calculates the hash value for a given memory address. Hash values are used to identify location of locked resources.

Author

Frane Jakelić

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

Parameters

blockMemoryAddress	integer representation of memory address, the hash value is calculated from this parameter.

Returns

integer containing the hash value of the passed memory address

5.108.2.18 AK_on_all_transactions_end()

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

Author

Ivan Pusic

5.108.2.19 AK_on_lock_release()

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

Author

Ivan Pusic

5.108.2.20 AK_on_observable_notify()

```
void AK_on_observable_notify (
            void * observer,
            void * observable,
            AK_ObservableType_Enum type )
```

Function for handling notify from some observable type.

Author

Ivan Pusic

Parameters

observer	Observer type
observable	Observable type
type	Type of observable who sent some notice

5.108.2.21 AK_on_transaction_end()

```
void AK_on_transaction_end (
            pthread_t transaction_thread )
```

Function for handling event when some transaction is finished.

Author

Ivan Pusic

Parameters

transaction_thread	Thread ID of transaction which is finished

5.108.2.22 AK_release_locks()

```
void AK_release_locks (
            AK_memoryAddresses_link addressesTmp,
            pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

Author

Frane Jakelić updated by Ivan Pusic

Parameters

adresses	linked list of memory addresses locked by the transaction.
transaction↩ Id	integer representation of transaction id.

5.108.2.23 AK_remove_transaction_thread()

```
int AK_remove_transaction_thread (
            pthread_t transaction_thread )
```

Function for deleting one of active threads from array of all active transactions threads.

Author

Ivan Pusic

Parameters

transaction_thread	Active thread to delete

Returns

Exit status (OK or NOT_OK)

5.108.2.24 AK_search_empty_link_for_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
            int blockAddress )
```

Function that searches for a empty link for new active block, helper method in case of address collision.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to empty location to store new active address

5.108.2.25 AK_search_existing_link_for_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
            int blockAddress )
```

Function that searches for a existing entry in hash list of active blocks.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to the existing hash list entry

5.108.2.26 AK_search_lock_entry_list_by_key()

```
AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (
            AK_transaction_elem_P Lockslist,
            int memoryAddress,
            pthread_t id )
```

Function that searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

NULL pointer if the element is not found otherwise it returns a pointer to the found element

5.108.2.27 AK_transaction_finished()

```
void AK_transaction_finished ( )
```

Function that is called when some transaction is finished.

Author

Ivan Pusic

5.108.2.28 AK_transaction_manager()

```
void AK_transaction_manager (
            command * commandArray,
            int lengthOfArray )
```

Function that receives all the data and gives an id to that data and starts a thread that executes the transaction.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray

5.108.2.29 AK_transaction_register_observer()

```
int AK_transaction_register_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for registering new observer of AK_observable_transaction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.108.2.30 AK_transaction_unregister_observer()

```
int AK_transaction_unregister_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for unregistering observer from AK_observable_transction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.108.2.31 handle_transaction_notify()

```
void handle_transaction_notify (
            AK_observer_lock * observer_lock )
```

Function for handling AK_observable_transaction notify. Function is associated to some observer instance.

Author

Ivan Pusic

Parameters

observer_lock	Observer type instance

5.109 trans/transaction.h File Reference

```
#include <pthread.h>
#include "../auxi/test.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)

  Function that calculates the 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)

  Function that searches for a existing entry in hash list of active blocks.

- AK_transaction_elem_P AK_search_empty_link_for_hook (int)

  Function that 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)

  Function that adds an element to the doubly linked list.

- int AK_delete_hash_entry_list (int)

  Function that 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)

  Function that 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)

  Function that 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)

  Function that, 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)

  Function that 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)

  Function that appends all addresses affected by the transaction.
- int AK_execute_commands (command *, int)

  Function 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 *)

  Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.
- void AK_transaction_manager (command *, int)

  Function that receives all the data and gives an id to that data and starts a thread that executes the transaction.
- TestResult **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 the lock is released.
- void AK_transaction_finished ()

  Function that is called when some transaction is finished.
- void AK_all_transactions_finished ()

  Function that is called when all transactions are finished.
- int AK_transaction_register_observer (AK_observable_transaction *, AK_observer *)

  Function for registering new observer of AK_observable_transaction type.
- int AK_transaction_unregister_observer (AK_observable_transaction *, AK_observer *)

  Function for unregistering observer from AK_observable_transction type.
- AK_observable_transaction * AK_init_observable_transaction ()

  Function for initialization of AK_observable_transaction type.
- AK_observer_lock * AK_init_observer_lock ()

  Function for initialization of AK_observer_lock type.

5.109.1 Detailed Description

Header file that contains data structures,functions and defines for the transaction execution

5.109.2 Enumeration Type Documentation

5.109.2.1 NoticeType

```
enum NoticeType
```

Enumeration which define notice types for transactions.

Author

Ivan Pusic

5.109.3 Function Documentation

5.109.3.1 AK_acquire_lock()

```
int AK_acquire_lock (
            int memoryAddress,
            int type,
            pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

Author

Frane Jakelić updated by Ivan Pusic

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

Parameters

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

Returns

OK or NOT_OK based on the success of the function.

Author

Frane Jakelić updated by Ivan Pusic

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

Parameters

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

Returns

OK or NOT_OK based on the success of the function.

########\n# Lock Granted after wait#\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */

##################\n# Lock Granted #\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */

5.109.3.2 AK_add_hash_entry_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
            int blockAddress,
            int type )
```

Function that adds an element to the doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
type	of lock issued to the provided memory address.

Returns

pointer to the newly created doubly linked element.

5.109.3.3 AK_add_lock()

```
AK_transaction_lock_elem_P AK_add_lock (
            AK_transaction_elem_P HashList,
            int type,
            pthread_t transactionId )
```

Function that adds an element to the locks doubly linked list.

Author

Frane Jakelić

Parameters

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

Returns

pointer to the newly created Locks doubly linked element.

5.109.3.4 AK_all_transactions_finished()

```
void AK_all_transactions_finished ( )
```

Function that is called when all transactions are finished.

Author

Ivan Pusic

5.109.3.5 AK_create_lock()

```
AK_transaction_lock_elem_P AK_create_lock (
            int blockAddress,
            int type,
            pthread_t transactionId )
```

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

Author

Frane Jakelić

Parameters

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

Returns

pointer to the newly created Locks doubly linked element.

5.109.3.6 AK_create_new_transaction_thread()

```
int AK_create_new_transaction_thread (
            AK_transaction_data * transaction_data )
```

Function for creating new thread. Function also adds thread ID to pthread_t array.

Author

Ivan Pusic

Parameters

transaction_data	Data for executing transaction

Returns

Exit status (OK or NOT_OK)

5.109.3.7 AK_delete_hash_entry_list()

```
int AK_delete_hash_entry_list (
            int blockAddress )
```

Function that deletes a specific element in the lockTable doubly linked list.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

integer OK or NOT_OK based on success of finding the specific element in the list.

5.109.3.8 AK_delete_lock_entry_list()

```
int AK_delete_lock_entry_list (
            int blockAddress,
            pthread_t id )
```

Function that deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

int OK or NOT_OK based on success of finding the specific element in the list.

5.109.3.9 AK_execute_commands()

```
int AK_execute_commands (
            command * commandArray,
            int lengthOfArray )
```

Function that is called in a separate thread that is responsible for acquiring locks, releasing them and finding the associated block addresses.

Author

Frane Jakelić updated by Ivan Pusic

Todo Check multithreading, check if it's working correctly

Parameters

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

Returns

ABORT or COMMIT based on the success of the function.

Author

Frane Jakelić updated by Ivan Pusic

Todo Check multithreading, check if it's working correctly

Parameters

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

Returns

ABORT or COMMIT based on the success of the function.

5.109.3.10 AK_execute_transaction()

```
void* AK_execute_transaction (
            void * params )
```

Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

data	transmitted to the thread from the main thread

5.109.3.11 AK_get_memory_blocks()

```
int AK_get_memory_blocks (
            char * tblName,
            AK_memoryAddresses_link addressList )
```

Function that appends all addresses affected by the transaction.

Author

Frane Jakelić

Parameters

addressList	pointer to the linked list where the addresses are stored.
tblName	table name used in the transaction

Returns

OK or NOT_OK based on the success of the function.

5.109.3.12 AK_handle_observable_transaction_action()

```
void AK_handle_observable_transaction_action (
            NoticeType * noticeType )
```

Function for handling action which is called from observable_transaction type.

Author

Ivan Pusic

Parameters

noticeType	Type of action (event)

5.109.3.13 AK_init_observable_transaction()

```
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

5.109.3.14 AK_init_observer_lock()

```
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

5.109.3.15 AK_isLock_waiting()

```
int AK_isLock_waiting (
            AK_transaction_elem_P lockHolder,
            int type,
            pthread_t transactionId,
            AK_transaction_lock_elem_P lock )
```

Function that, based on the parameters, puts an transaction action in waiting phase or let's the transaction do it's actions.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

lockHolder	pointer to the hash list entry that is entitled to the specific memory address.
type	of lock issued to the provided memory address.
transaction↩ Id	integer representation of transaction id.
lock	pointer to the lock element that is being tested.

Returns

int PASS_LOCK_QUEUE or WAIT_FOR_UNLOCK based on the rules described inside the function.

5.109.3.16 AK_lock_released()

```
void AK_lock_released ( )
```

Function which is called when the lock is released.

Author

Ivan Pusic

5.109.3.17 AK_memory_block_hash()

```
int AK_memory_block_hash (
            int blockMemoryAddress )
```

Function that calculates the hash value for a given memory address. Hash values are used to identify location of locked resources.

Author

Frane Jakelić

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

Parameters

blockMemoryAddress	integer representation of memory address, the hash value is calculated from this parameter.

Returns

integer containing the hash value of the passed memory address

Author

Frane Jakelić

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

Parameters

blockMemoryAddress	integer representation of memory address, the hash value is calculated from this parameter.

Returns

integer containing the hash value of the passed memory address

5.109.3.18 AK_on_all_transactions_end()

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

Author

Ivan Pusic

5.109.3.19 AK_on_lock_release()

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

Author

Ivan Pusic

5.109.3.20 AK_on_observable_notify()

```
void AK_on_observable_notify (
            void * observer,
            void * observable,
            AK_ObservableType_Enum type )
```

Function for handling notify from some observable type.

Author

Ivan Pusic

Parameters

observer	Observer type
observable	Observable type
type	Type of observable who sent some notice

5.109.3.21 AK_on_transaction_end()

```
void AK_on_transaction_end (
            pthread_t transaction_thread )
```

Function for handling event when some transaction is finished.

Author

Ivan Pusic

Parameters

transaction_thread	Thread ID of transaction which is finished

5.109.3.22 AK_release_locks()

```
void AK_release_locks (
            AK_memoryAddresses_link addressesTmp,
            pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

Author

Frane Jakelić updated by Ivan Pusic

Parameters

adresses	linked list of memory addresses locked by the transaction.
transaction↩ Id	integer representation of transaction id.

5.109.3.23 AK_remove_transaction_thread()

```
int AK_remove_transaction_thread (
            pthread_t transaction_thread )
```

Function for deleting one of active threads from array of all active transactions threads.

Author

Ivan Pusic

Parameters

transaction_thread	Active thread to delete

Returns

Exit status (OK or NOT_OK)

5.109.3.24 AK_search_empty_link_for_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
            int blockAddress )
```

Function that searches for a empty link for new active block, helper method in case of address collision.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to empty location to store new active address

5.109.3.25 AK_search_existing_link_for_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
            int blockAddress )
```

Function that searches for a existing entry in hash list of active blocks.

Author

Frane Jakelić

Parameters

blockAddress	integer representation of memory address.

Returns

pointer to the existing hash list entry

5.109.3.26 AK_search_lock_entry_list_by_key()

```
AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (
            AK_transaction_elem_P Lockslist,
            int memoryAddress,
            pthread_t id )
```

Function that searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

Author

Frane Jakelić

Parameters

memoryAddress	integer representation of memory address.
id	integer representation of transaction id.

Returns

NULL pointer if the element is not found otherwise it returns a pointer to the found element

5.109.3.27 AK_transaction_finished()

```
void AK_transaction_finished ( )
```

Function that is called when some transaction is finished.

Author

Ivan Pusic

5.109.3.28 AK_transaction_manager()

```
void AK_transaction_manager (
            command * commandArray,
            int lengthOfArray )
```

Function that receives all the data and gives an id to that data and starts a thread that executes the transaction.

Author

Frane Jakelić updated by Ivan Pusic

Parameters

commandArray	array filled with commands that need to be secured using transactions
lengthOfArray	length of commandArray

5.109.3.29 AK_transaction_register_observer()

```
int AK_transaction_register_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for registering new observer of AK_observable_transaction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.109.3.30 AK_transaction_unregister_observer()

```
int AK_transaction_unregister_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for unregistering observer from AK_observable_transction type.

Author

Ivan Pusic

Parameters

observable_transaction	Observable type instance
observer	Observer instance

Returns

Exit status (OK or NOT_OK)

5.109.3.31 handle_transaction_notify()

```
void handle_transaction_notify (
            AK_observer_lock * observer_lock )
```

Function for handling AK_observable_transaction notify. Function is associated to some observer instance.

Author

Ivan Pusic

Parameters

observer_lock	Observer type instance

Index

dictionary, 11
    hash, 11
    key, 11
    size, 12
    val, 12
_file_metadata, 12
_line_status_
    iniparser.c, 82
_notifyDetails, 12

aggregation.c
    AK_agg_input_add, 367
    AK_agg_input_add_to_beginning, 367
    AK_agg_input_fix, 368
    AK_agg_input_init, 368
    AK_aggregation, 369
    AK_aggregation_test, 370
    AK_header_size, 370
    AK_search_unsorted, 370
aggregation.h
    AK_agg_input_add, 372
    AK_agg_input_add_to_beginning, 373
    AK_agg_input_fix, 373
    AK_agg_input_init, 374
    AK_aggregation, 374
    AK_aggregation_test, 375
    AK_header_size, 375
AK_acquire_lock
    transaction.c, 527
    transaction.h, 543
AK_add_hash_entry_list
    transaction.c, 528
    transaction.h, 544
AK_add_lock
    transaction.c, 528
    transaction.h, 545
AK_add_reference
    reference.c, 444
    reference.h, 450
AK_add_succesor
    auxiliary.h, 48
AK_add_to_bitmap_index
    bitmap.c, 216
    bitmap.h, 223
AK_add_to_redolog
    redo_log.c, 364
AK_add_to_redolog_select
    redo_log.c, 364
AK_add_user_to_group
    privileges.c, 483
    privileges.h, 495
AK_add_vertex
    auxiliary.h, 48
AK_agg_input, 13
AK_agg_input_add
    aggregation.c, 367
    aggregation.h, 372
AK_agg_input_add_to_beginning
    aggregation.c, 367
    aggregation.h, 373
AK_agg_input_fix
    aggregation.c, 368
    aggregation.h, 373
AK_agg_input_init
    aggregation.c, 368
    aggregation.h, 374
AK_agg_value, 13
AK_aggregation
    aggregation.c, 369
    aggregation.h, 374
AK_aggregation_test
    aggregation.c, 370
    aggregation.h, 375
AK_all_transactions_finished
    transaction.c, 529
    transaction.h, 545
AK_allocate_block_activity_modes
    dbman.c, 144
AK_allocate_blocks
    dbman.c, 144
    dbman.h, 164
AK_allocation_set_mode
    dbman.h, 163
AK_ALLOCATION_TABLE_SIZE
    dbman.h, 163
AK_allocationbit
    dbman.h, 179
AK_allocationtable_dump
    dbman.c, 144
    dbman.h, 164
AK_archive_log
    archive_log.h, 358
AK_bitmap_test
    bitmap.c, 217
    bitmap.h, 224
AK_block, 14
AK_block_activity, 14
AK_block_sort
    filesort.h, 209

AK_blocktable, 15
AK_blocktable_dump
    dbman.c, 145
    dbman.h, 165
AK_blocktable_flush
    dbman.c, 145
    dbman.h, 165
AK_blocktable_get
    dbman.c, 145
    dbman.h, 165
AK_btree_create
    btree.c, 230
    btree.h, 232
AK_btree_search_delete
    btree.c, 231
    btree.h, 232
AK_cache_AK_malloc
    memoman.c, 299
    memoman.h, 310
AK_cache_block
    memoman.c, 300
    memoman.h, 310
AK_cache_result
    memoman.c, 300
    memoman.h, 311
AK_calloc
    mempro.c, 100
    mempro.h, 116
AK_change_hash_info
    hash.c, 234
    hash.h, 240
AK_chars_num_from_number
    auxiliary.h, 48
AK_check_arithmetic_statement
    expression_check.c, 379
    expression_check.h, 382
AK_check_attributes
    redo_log.c, 365
AK_check_constraint
    check_constraint.c, 430
    check_constraint.h, 433
AK_check_constraint_name
    constraint_names.c, 435
    constraint_names.h, 436
AK_check_constraint_not_null
    nnull.c, 438
    nnull.h, 441
AK_check_constraint_test
    check_constraint.c, 431
    check_constraint.h, 433
AK_check_constraints
    theta_join.c, 412
    theta_join.h, 415
AK_check_folder_blobs
    blobs.c, 180
    blobs.h, 185
AK_check_for_writes
    mempro.c, 101
    mempro.h, 116
AK_check_function_arguments
    function.c, 469
    function.h, 475
AK_check_function_arguments_type
    function.c, 469
    function.h, 476
AK_check_group_privilege
    privileges.c, 484
    privileges.h, 495
AK_check_if_row_satisfies_expression
    expression_check.c, 380
    expression_check.h, 383
AK_check_privilege
    privileges.c, 484
    privileges.h, 496
AK_check_redo_log_select
    redo_log.c, 365
AK_check_regex_expression
    expression_check.c, 380
    expression_check.h, 384
AK_check_regex_operator_expression
    expression_check.c, 381
    expression_check.h, 384
AK_check_tables_scheme
    table.c, 273
    table.h, 287
AK_check_user_privilege
    privileges.c, 485
    privileges.h, 496
AK_check_view_name
    view.c, 519
AK_command
    command.c, 421
    command.h, 422
AK_command_recovery_struct, 15
AK_command_struct, 16
AK_compare
    rel_eq_assoc.c, 325
    rel_eq_assoc.h, 327
AK_concat
    blobs.c, 180
    blobs.h, 185
AK_constraint_between_test
    between.c, 423
    between.h, 427
AK_constraint_names_test
    constraint_names.c, 435
    constraint_names.h, 437
AK_CONSTRAINTS_DEFAULT
    constants.h, 68
AK_CONSTRAINTS_FOREIGN_KEY
    constants.h, 68
AK_CONSTRAINTS_INDEX
    constants.h, 69
AK_CONSTRAINTS_PRIMARY_KEY
    constants.h, 69
AK_convert_type

auxiliary.h, 49
AK_copy_block_projection
projection.c, 398
projection.h, 403
AK_copy_blocks_join
nat_join.c, 388
nat_join.h, 391
AK_copy_header
dbman.c, 146
dbman.h, 165
AK_create_block_header
projection.c, 398
projection.h, 404
AK_create_hash_index
hash.c, 234
hash.h, 241
AK_create_header
dbman.c, 146
dbman.h, 166
AK_create_header_name
projection.c, 399
projection.h, 405
AK_create_Index
bitmap.c, 217
bitmap.h, 224
AK_create_Index_Table
bitmap.c, 218
bitmap.h, 225
AK_create_join_block_header
nat_join.c, 389
nat_join.h, 392
AK_create_lock
transaction.c, 529
transaction.h, 545
AK_create_new_transaction_thread
transaction.c, 529
transaction.h, 546
AK_create_table
table.c, 273
table.h, 287
AK_create_table_struct, 16
AK_create_test_tables
test.c, 134
test.h, 138
AK_create_theta_join_header
theta_join.c, 413
theta_join.h, 416
AK_db_cache, 17
AK_dbg_messg
debug.c, 70
debug.h, 71
AK_deallocate_search_result
filesearch.c, 205
filesearch.h, 207
AK_debmod_calloc
mempro.c, 101
mempro.h, 117
AK_debmod_d
mempro.c, 102
mempro.h, 117
AK_debmod_die
mempro.c, 102
mempro.h, 118
AK_debmod_dv
mempro.c, 102
mempro.h, 118
AK_debmod_enter_critical_sec
mempro.c, 103
mempro.h, 119
AK_debmod_free
mempro.c, 103
mempro.h, 119
AK_debmod_fstack_pop
mempro.c, 104
mempro.h, 119
AK_debmod_fstack_push
mempro.c, 104
mempro.h, 120
AK_debmod_func_add
mempro.c, 105
mempro.h, 120
AK_debmod_func_get_name
mempro.c, 105
mempro.h, 121
AK_debmod_func_id
mempro.c, 106
mempro.h, 121
AK_debmod_function_current
mempro.c, 106
mempro.h, 122
AK_debmod_function_epilogue
mempro.c, 107
mempro.h, 122
AK_debmod_function_prologue
mempro.c, 107
mempro.h, 123
AK_debmod_init
mempro.c, 108
mempro.h, 123
AK_debmod_leave_critical_sec
mempro.c, 108
mempro.h, 123
AK_debmod_log_memory_alloc
mempro.c, 108
mempro.h, 124
AK_debmod_print_function_use
mempro.c, 109
mempro.h, 124
AK_debmod_state, 17
AK_Delete_All_elementsAd
index.c, 247
index.h, 256
AK_delete_bitmap_index
bitmap.c, 219
bitmap.h, 226
AK_delete_block

dbman.c, 147
dbman.h, 167
AK_delete_constraint_between
between.c, 424
between.h, 427
AK_delete_constraint_not_null
nnull.c, 438
nnull.h, 441
AK_delete_constraint_unique
unique.c, 458
unique.h, 461
AK_Delete_elementAd
index.c, 247
index.h, 256
AK_delete_extent
dbman.c, 147
dbman.h, 167
AK_delete_hash_entry_list
transaction.c, 530
transaction.h, 546
AK_delete_in_hash_index
hash.c, 235
hash.h, 241
AK_Delete_L3
auxiliary.h, 50
AK_delete_lock_entry_list
transaction.c, 530
transaction.h, 547
AK_delete_row
fileio.c, 189
fileio.h, 195
reference.h, 450
AK_delete_row_by_id
fileio.c, 189
fileio.h, 195
AK_delete_row_from_block
fileio.c, 189
fileio.h, 196
AK_delete_segment
dbman.c, 148
dbman.h, 168
AK_delete_update_segment
fileio.c, 190
fileio.h, 196
AK_DeleteAll_L3
auxiliary.h, 50
AK_destroy_critical_section
auxiliary.h, 50
AK_determine_header_type
projection.c, 400
projection.h, 405
AK_difference
difference.c, 376
difference.h, 377
AK_drop
drop.c, 464
drop.h, 467
AK_drop_help_function
drop.c, 465
AK_drop_test
drop.c, 465
drop.h, 467
AK_elem_hash_value
hash.c, 235
hash.h, 242
AK_End_L2
auxiliary.h, 51
AK_enter_critical_section
auxiliary.h, 51
AK_execute_commands
transaction.c, 531
transaction.h, 547
AK_execute_rel_eq
query_optimization.c, 319
query_optimization.h, 322
AK_execute_transaction
transaction.c, 531
transaction.h, 548
AK_files_test
files.c, 201
files.h, 203
AK_filesearch_test
filesearch.c, 205
filesearch.h, 208
AK_find_AK_free_space
memoman.c, 301
memoman.h, 311
AK_find_available_result_block
memoman.c, 301
memoman.h, 312
AK_find_delete_in_hash_index
hash.c, 235
hash.h, 242
AK_find_in_hash_index
hash.c, 236
hash.h, 243
AK_find_table_address
between.c, 424
between.h, 428
AK_First_L2
auxiliary.h, 52
AK_flush_cache
memoman.c, 301
memoman.h, 312
AK_folder_exists
blobs.c, 181
blobs.h, 185
AK_fread
mempro.c, 109
AK_free
mempro.c, 110
mempro.h, 125
AK_function_add
function.c, 470
function.h, 476
AK_function_arguments_add

function.c, 470
function.h, 477
AK_function_arguments_remove_by_obj_id
function.c, 471
function.h, 477
AK_function_change_return_type
function.c, 471
function.h, 478
AK_function_remove_by_name
function.c, 472
function.h, 478
AK_function_remove_by_obj_id
function.c, 472
function.h, 479
AK_function_rename
function.c, 473
function.h, 479
AK_function_test
function.c, 473
function.h, 480
AK_fwrite
mempro.c, 110
AK_generate_result_id
memoman.c, 302
memoman.h, 312
AK_get_allocation_set
dbman.c, 148
dbman.h, 168
AK_get_array_perms
auxiliary.h, 52
AK_get_attr_index
table.c, 274
table.h, 288
AK_get_attr_name
table.c, 274
table.h, 288
AK_get_Attribute
bitmap.c, 220
bitmap.h, 227
AK_get_attribute
bitmap.c, 219
bitmap.h, 226
AK_get_block
memoman.c, 302
memoman.h, 313
AK_get_column
table.c, 275
table.h, 289
AK_get_extent
dbman.c, 149
dbman.h, 169
AK_Get_First_elementAd
index.c, 248
index.h, 256
AK_get_function_obj_id
function.c, 474
function.h, 480
AK_get_hash_info
hash.c, 236
hash.h, 243
AK_get_header
table.c, 275
table.h, 289
AK_get_header_number
filesort.h, 210
AK_get_id
id.c, 213
id.h, 214
AK_get_index_addresses
memoman.c, 303
memoman.h, 313
AK_get_index_header
index.c, 248
AK_get_index_num_records
index.c, 249
index.h, 257
AK_get_index_segment_addresses
memoman.c, 303
memoman.h, 314
AK_get_index_tuple
index.c, 249
index.h, 257
AK_get_insert_header
insert.h, 481
AK_Get_Last_elementAd
index.c, 250
index.h, 258
AK_get_memory_blocks
transaction.c, 532
transaction.h, 548
AK_Get_Next_elementAd
index.c, 250
index.h, 258
AK_get_nth_main_bucket_add
hash.c, 237
hash.h, 244
AK_get_num_of_tuples
filesort.h, 210
AK_get_num_records
table.c, 277
table.h, 290
AK_get_operator
projection.c, 400
projection.h, 406
AK_Get_Position_Of_elementAd
index.c, 251
index.h, 259
AK_Get_Previous_elementAd
index.c, 251
index.h, 259
AK_get_reference
reference.c, 445
reference.h, 451
AK_get_rel_exp
view.c, 520
AK_get_row

table.c, 277
table.h, 290
AK_get_segment_addresses
memoman.c, 304
memoman.h, 314
AK_get_segment_addresses_internal
memoman.c, 304
memoman.h, 315
AK_get_system_table_address
memoman.c, 305
AK_get_table_addresses
memoman.c, 305
memoman.h, 315
AK_get_table_atribute_types
test.c, 134
test.h, 138
AK_get_table_id
id.c, 213
AK_get_table_obj_id
table.c, 278
table.h, 291
AK_get_timestamp
archive_log.h, 358
AK_get_total_headers
filesort.h, 211
AK_get_tuple
table.c, 278
table.h, 291
AK_get_view_obj_id
view.c, 520
AK_get_view_query
view.c, 520
AK_GetNth_L2
auxiliary.h, 53
AK_grant_privilege_group
privileges.c, 485
privileges.h, 497
AK_grant_privilege_user
privileges.c, 486
privileges.h, 497
AK_group_add
privileges.c, 486
privileges.h, 498
AK_group_get_id
privileges.c, 487
privileges.h, 498
AK_group_remove_by_name
privileges.c, 487
privileges.h, 499
AK_group_rename
privileges.c, 487
privileges.h, 499
AK_GUID
blobs.c, 181
blobs.h, 185
AK_handle_observable_transaction_action
transaction.c, 532
transaction.h, 549
AK_hash_test
hash.c, 237
hash.h, 244
AK_header, 18
AK_header_size
aggregation.c, 370
aggregation.h, 375
AK_id_test
id.c, 213
id.h, 215
AK_if_exist
drop.c, 465
drop.h, 468
AK_If_ExistOp
bitmap.c, 220
bitmap.h, 227
AK_increase_extent
dbman.c, 149
dbman.h, 169
AK_index_table_exist
index.c, 252
index.h, 260
AK_index_test
index.c, 252
index.h, 260
AK_init_allocation_table
dbman.c, 150
dbman.h, 170
AK_init_block
dbman.c, 150
dbman.h, 170
AK_init_critical_section
auxiliary.h, 55
AK_init_db_file
dbman.c, 151
dbman.h, 170
AK_init_disk_manager
dbman.c, 151
dbman.h, 171
AK_Init_L3
auxiliary.h, 55
AK_init_new_extent
memoman.c, 305
memoman.h, 316
AK_init_observable
observable.c, 129
observable.h, 131
AK_init_observable_transaction
transaction.c, 532
transaction.h, 549
AK_init_observer
observable.c, 130
observable.h, 132
AK_init_observer_lock
transaction.c, 533
transaction.h, 549
AK_init_system_catalog
dbman.c, 152

dbman.h, 171
AK_init_system_tables_catalog
dbman.c, 152
dbman.h, 171
AK_initialize_new_index_segment
files.c, 201
files.h, 203
AK_initialize_new_segment
files.c, 201
files.h, 203
reference.h, 451
AK_InitializelistAd
index.c, 252
index.h, 260
AK_insert
insert.h, 482
AK_insert_bucket_to_block
hash.c, 238
hash.h, 244
AK_insert_entry
dbman.c, 153
dbman.h, 173
AK_insert_in_hash_index
hash.c, 238
hash.h, 245
AK_Insert_New_Element
fileio.c, 190
fileio.h, 197
reference.h, 452
AK_Insert_New_Element_For_Update
fileio.c, 191
fileio.h, 197
reference.h, 452
AK_Insert_NewelementAd
index.c, 253
index.h, 261
AK_insert_row
fileio.c, 192
fileio.h, 198
reference.h, 453
AK_insert_row_to_block
fileio.c, 192
fileio.h, 199
AK_InsertAfter_L2
auxiliary.h, 55
AK_InsertAtBegin_L3
auxiliary.h, 56
AK_InsertAtEnd_L3
auxiliary.h, 56
AK_InsertBefore_L2
auxiliary.h, 57
AK_intersect
intersect.c, 385
intersect.h, 387
AK_IsEmpty_L2
auxiliary.h, 57
AK_isLock_waiting
transaction.c, 533
transaction.h, 550
AK_join
nat_join.c, 389
nat_join.h, 392
AK_leave_critical_section
auxiliary.h, 58
AK_lo_export
blobs.c, 181
blobs.h, 186
AK_lo_import
blobs.c, 182
blobs.h, 186
AK_lo_test
blobs.c, 182
blobs.h, 186
AK_lo_unlink
blobs.c, 182
blobs.h, 187
AK_load_chosen_log
recovery.c, 360
AK_load_latest_log
recovery.c, 360
AK_lock_released
transaction.c, 534
transaction.h, 550
AK_malloc
mempro.c, 111
mempro.h, 125
AK_mem_block, 19
AK_mem_block_modify
memoman.c, 306
memoman.h, 316
AK_memoman_init
memoman.c, 306
memoman.h, 317
AK_memory_block_hash
transaction.c, 534
transaction.h, 551
AK_mempro_test
mempro.c, 111
mempro.h, 126
AK_memset_int
dbman.c, 154
dbman.h, 173
AK_merge_block_join
nat_join.c, 390
nat_join.h, 393
AK_mkdir
blobs.c, 183
blobs.h, 187
AK_new_extent
dbman.c, 154
dbman.h, 174
AK_new_segment
dbman.c, 155
dbman.h, 175
AK_Next_L2
auxiliary.h, 58

AK_nnull_constraint_test
nnull.c, 439
nnull.h, 442
AK_num_attr
table.c, 279
table.h, 292
AK_num_index_attr
index.c, 253
index.h, 261
AK_observable_test
observable.c, 130
observable.h, 132
AK_on_all_transactions_end
transaction.c, 535
transaction.h, 551
AK_on_lock_release
transaction.c, 535
transaction.h, 552
AK_on_observable_notify
transaction.c, 535
transaction.h, 552
AK_on_transaction_end
transaction.c, 536
transaction.h, 552
AK_op_difference_test
difference.c, 377
difference.h, 378
AK_op_intersect_test
intersect.c, 386
intersect.h, 387
AK_op_join_test
nat_join.c, 390
nat_join.h, 393
AK_op_product_test
product.c, 394
product.h, 396
AK_op_projection_test
projection.c, 400
projection.h, 406
AK_op_rename_test
table.c, 279
table.h, 292
AK_op_selection_test
selection.c, 409
selection.h, 411
AK_op_selection_test_pattern
selection.c, 409
selection.h, 411
AK_op_theta_join_test
theta_join.c, 413
theta_join.h, 416
AK_op_union_test
union.c, 418
union.h, 419
AK_operand, 19
AK_perform_operation
projection.c, 401
projection.h, 407
AK_pop_from_stack
auxiliary.h, 59
AK_Previous_L2
auxiliary.h, 59
AK_print_active_functions
mempro.c, 111
mempro.h, 126
AK_print_Att_Test
bitmap.c, 220
bitmap.h, 227
AK_print_block
dbman.c, 156
dbman.h, 175
AK_print_constraints
between.c, 425
AK_print_function_use
mempro.c, 111
mempro.h, 126
AK_print_function_uses
mempro.c, 112
mempro.h, 127
AK_print_Header_Test
bitmap.c, 221
bitmap.h, 228
AK_print_index_table
index.c, 254
index.h, 262
AK_print_optimized_query
query_optimization.c, 320
query_optimization.h, 323
AK_print_rel_eq_assoc
rel_eq_assoc.c, 325
rel_eq_assoc.h, 328
AK_print_rel_eq_comut
rel_eq_comut.c, 329
rel_eq_comut.h, 332
AK_print_rel_eq_projection
rel_eq_projection.c, 334
rel_eq_projection.h, 340
AK_print_rel_eq_selection
rel_eq_selection.c, 346
rel_eq_selection.h, 351
AK_print_row
table.c, 279
table.h, 293
AK_print_row_spacer
table.c, 280
table.h, 293
AK_print_row_spacer_to_file
table.c, 280
table.h, 294
AK_print_row_to_file
table.c, 281
table.h, 294
AK_print_table
table.c, 281
table.h, 295
AK_print_table_to_file

table.c, 282
table.h, 295
AK_printout_redolog
redo_log.c, 365
AK_privileges_test
privileges.c, 488
privileges.h, 499
AK_product
product.c, 394
product.h, 396
AK_product_procedure
product.c, 395
product.h, 397
AK_projection
projection.c, 401
projection.h, 407
AK_push_to_stack
auxiliary.h, 59
AK_query_mem, 20
AK_query_mem_AK_free
memoman.c, 306
memoman.h, 317
AK_query_mem_AK_malloc
memoman.c, 307
memoman.h, 317
AK_query_mem_dict, 20
AK_query_mem_lib, 21
AK_query_mem_result, 21
AK_query_optimization
query_optimization.c, 320
query_optimization.h, 323
AK_query_optimization_test
query_optimization.c, 321
query_optimization.h, 324
AK_read_block
dbman.c, 156
dbman.h, 175
AK_read_block_for_testing
dbman.c, 156
dbman.h, 176
AK_read_constraint_between
between.c, 425
between.h, 428
AK_read_constraint_not_null
nnull.c, 439
nnull.h, 442
AK_read_constraint_unique
unique.c, 459
unique.h, 462
AK_realloc
mempro.c, 112
mempro.h, 127
AK_recover_archive_log
recovery.c, 361
AK_recover_operation
recovery.c, 361
AK_recovery_insert_row
recovery.c, 362
AK_recovery_test
recovery.c, 362
AK_recovery_tokenize
recovery.c, 362
AK_redo_log, 22
AK_redo_log_AK_malloc
memoman.c, 307
memoman.h, 318
AK_ref_item, 23
AK_reference_check_attribute
reference.c, 445
reference.h, 454
AK_reference_check_entry
reference.c, 446
reference.h, 454
AK_reference_check_if_update_needed
reference.c, 446
reference.h, 455
AK_reference_check_restricion
reference.c, 446
reference.h, 455
AK_reference_test
reference.c, 447
reference.h, 456
AK_reference_update
reference.c, 447
reference.h, 456
AK_refresh_cache
memoman.c, 307
memoman.h, 318
AK_register_system_tables
dbman.c, 157
dbman.h, 176
AK_rel_eq_assoc
rel_eq_assoc.c, 326
rel_eq_assoc.h, 328
AK_rel_eq_assoc_test
rel_eq_assoc.c, 326
rel_eq_assoc.h, 328
AK_rel_eq_can_commute
rel_eq_projection.c, 335
rel_eq_projection.h, 340
AK_rel_eq_collect_cond_attributes
rel_eq_projection.c, 335
rel_eq_projection.h, 341
AK_rel_eq_commute_with_theta_join
rel_eq_comut.c, 330
rel_eq_comut.h, 332
AK_rel_eq_comut
rel_eq_comut.c, 330
rel_eq_comut.h, 333
AK_rel_eq_comut_test
rel_eq_comut.c, 331
rel_eq_comut.h, 333
AK_rel_eq_cond_attributes
rel_eq_selection.c, 346
rel_eq_selection.h, 351
AK_rel_eq_get_atrributes_char

rel_eq_selection.c, 347
rel_eq_selection.h, 352
AK_rel_eq_get_attributes
rel_eq_projection.c, 336
rel_eq_projection.h, 341
AK_rel_eq_is_attr_subset
rel_eq_selection.c, 347
rel_eq_selection.h, 354
AK_rel_eq_is_subset
rel_eq_projection.c, 336
rel_eq_projection.h, 342
AK_rel_eq_projection
rel_eq_projection.c, 337
rel_eq_projection.h, 343
AK_rel_eq_projection_attributes
rel_eq_projection.c, 338
rel_eq_projection.h, 344
AK_rel_eq_projection_test
rel_eq_projection.c, 338
rel_eq_projection.h, 344
AK_rel_eq_remove_duplicates
rel_eq_projection.c, 339
rel_eq_projection.h, 345
AK_rel_eq_selection
rel_eq_selection.c, 348
rel_eq_selection.h, 355
AK_rel_eq_selection_test
rel_eq_selection.c, 348
rel_eq_selection.h, 355
AK_rel_eq_share_attributes
rel_eq_selection.c, 349
rel_eq_selection.h, 355
AK_rel_eq_split_condition
rel_eq_selection.c, 349
rel_eq_selection.h, 356
AK_release_locks
transaction.c, 536
transaction.h, 553
AK_release_oldest_cache_block
memoman.c, 308
memoman.h, 318
AK_remove_all_users_from_group
privileges.c, 488
privileges.h, 500
AK_remove_substring
projection.c, 402
projection.h, 408
AK_remove_transaction_thread
transaction.c, 536
transaction.h, 553
AK_remove_user_from_all_groups
privileges.c, 489
privileges.h, 500
AK_rename
table.c, 282
table.h, 295
AK_replace_wild_card
expression_check.c, 381
AK_reset_block
filesort.h, 211
AK_results, 23
AK_Retrieve_L2
auxiliary.h, 60
AK_revoke_all_privileges_group
privileges.c, 489
privileges.h, 501
AK_revoke_all_privileges_user
privileges.c, 490
privileges.h, 501
AK_revoke_privilege_group
privileges.c, 490
privileges.h, 502
AK_revoke_privilege_user
privileges.c, 491
privileges.h, 502
AK_search_empty_link
auxiliary.h, 60
AK_search_empty_link_for_hook
transaction.c, 537
transaction.h, 554
AK_search_empty_stack_link
auxiliary.h, 61
AK_search_existing_link_for_hook
transaction.c, 537
transaction.h, 554
AK_search_in_stack
auxiliary.h, 61
AK_search_lock_entry_list_by_key
transaction.c, 538
transaction.h, 554
AK_search_unsorted
aggregation.c, 370
filesearch.c, 205
filesearch.h, 208
AK_search_vertex
auxiliary.h, 62
AK_select
select.c, 506
select.h, 507
AK_select_test
select.c, 506
select.h, 508
AK_selection
reference.h, 456
selection.c, 409
selection.h, 411
AK_selection_op_rename
selection.c, 410
AK_sequence_add
sequence.c, 263
sequence.h, 268
AK_sequence_current_value
sequence.c, 264
sequence.h, 268
AK_sequence_get_id
sequence.c, 264

sequence.h, 269
AK_sequence_modify
sequence.c, 264
sequence.h, 269
AK_sequence_next_value
sequence.c, 265
sequence.h, 270
AK_sequence_remove
sequence.c, 266
sequence.h, 270
AK_sequence_rename
sequence.c, 266
sequence.h, 270
AK_sequence_test
sequence.c, 266
sequence.h, 271
AK_set_check_constraint
check_constraint.c, 431
check_constraint.h, 433
AK_set_constraint_between
between.c, 426
between.h, 429
AK_set_constraint_not_null
nnull.c, 440
nnull.h, 443
AK_set_constraint_unique
unique.c, 460
unique.h, 462
AK_Size_L2
auxiliary.h, 62
AK_sort_segment
filesort.h, 212
AK_split_path_file
blobs.c, 183
blobs.h, 187
AK_strcmp
auxiliary.h, 62
AK_synchronization_info, 24
AK_table_empty
table.c, 283
table.h, 296
AK_table_exist
table.c, 283
AK_table_test
table.c, 283
table.h, 296
AK_tarjan
auxiliary.h, 63
AK_temp_create_table
table.c, 284
table.h, 297
AK_test_command
command.c, 421
command.h, 422
AK_test_get_view_data
view.c, 521
AK_theta_join
theta_join.c, 414
theta_join.h, 416
AK_thread_safe_block_access_test
dbman.c, 158
dbman.h, 177
AK_transaction_finished
transaction.c, 538
transaction.h, 555
AK_transaction_manager
transaction.c, 538
transaction.h, 555
AK_transaction_register_observer
transaction.c, 539
transaction.h, 556
AK_transaction_unregister_observer
transaction.c, 539
transaction.h, 556
AK_trigger_add
trigger.c, 509
trigger.h, 514
AK_trigger_edit
trigger.c, 509
trigger.h, 514
AK_trigger_get_conditions
trigger.c, 510
trigger.h, 515
AK_trigger_get_id
trigger.c, 510
trigger.h, 516
AK_trigger_remove_by_name
trigger.c, 511
trigger.h, 516
AK_trigger_remove_by_obj_id
trigger.c, 511
trigger.h, 517
AK_trigger_rename
trigger.c, 512
trigger.h, 517
AK_trigger_save_conditions
trigger.c, 512
trigger.h, 518
AK_trigger_test
trigger.c, 513
trigger.h, 518
AK_tuple_dict, 24
AK_tuple_to_string
table.c, 284
table.h, 297
AK_type_size
auxiliary.h, 63
AK_union
union.c, 418
union.h, 419
AK_unique_test
unique.c, 460
unique.h, 463
AK_update
bitmap.c, 221
bitmap.h, 228

AK_update_bucket_in_block
hash.c, 239
hash.h, 245
AK_Update_Existing_Element
fileio.c, 193
reference.h, 457
AK_update_row
fileio.c, 193
fileio.h, 199
reference.h, 457
AK_update_row_from_block
fileio.c, 194
fileio.h, 199
AK_user_add
privileges.c, 491
privileges.h, 503
AK_user_check_pass
privileges.c, 492
privileges.h, 504
AK_user_get_id
privileges.c, 492
privileges.h, 504
AK_user_remove_by_name
privileges.c, 492
AK_user_rename
privileges.c, 493
privileges.h, 505
AK_view_add
view.c, 521
AK_view_change_query
view.c, 522
AK_view_remove_by_name
view.c, 522
AK_view_remove_by_obj_id
view.c, 523
AK_view_rename
view.c, 523
AK_view_test
view.c, 523
AK_write_block
bitmap.h, 229
dbman.c, 158
dbman.h, 178
AK_write_block_for_testing
dbman.c, 159
dbman.h, 178
AK_write_protect
mempro.c, 113
mempro.h, 127
AK_write_unprotect
mempro.c, 113
mempro.h, 128
archive_log.h
AK_archive_log, 358
AK_get_timestamp, 358
auxi/auxiliary.c, 45
auxi/auxiliary.h, 45
auxi/constants.h, 64
auxi/debug.c, 69
auxi/debug.h, 70
auxi/dictionary.c, 72
auxi/dictionary.h, 76
auxi/iniparser.c, 80
auxi/iniparser.h, 90
auxi/mempro.c, 99
auxi/mempro.h, 114
auxi/observable.c, 128
auxi/observable.h, 131
auxi/test.c, 132
auxiliary.h
AK_add_succesor, 48
AK_add_vertex, 48
AK_chars_num_from_number, 48
AK_convert_type, 49
AK_Delete_L3, 50
AK_DeleteAll_L3, 50
AK_destroy_critical_section, 50
AK_End_L2, 51
AK_enter_critical_section, 51
AK_First_L2, 52
AK_get_array_perms, 52
AK_GetNth_L2, 53
AK_init_critical_section, 55
AK_Init_L3, 55
AK_InsertAfter_L2, 55
AK_InsertAtBegin_L3, 56
AK_InsertAtEnd_L3, 56
AK_InsertBefore_L2, 57
AK_IsEmpty_L2, 57
AK_leave_critical_section, 58
AK_Next_L2, 58
AK_pop_from_stack, 59
AK_Previous_L2, 59
AK_push_to_stack, 59
AK_Retrieve_L2, 60
AK_search_empty_link, 60
AK_search_empty_stack_link, 61
AK_search_in_stack, 61
AK_search_vertex, 62
AK_Size_L2, 62
AK_strcmp, 62
AK_tarjan, 63
AK_type_size, 63
testMode, 64

between.c
AK_constraint_between_test, 423
AK_delete_constraint_between, 424
AK_find_table_address, 424
AK_print_constraints, 425
AK_read_constraint_between, 425
AK_set_constraint_between, 426
between.h
AK_constraint_between_test, 427
AK_delete_constraint_between, 427
AK_find_table_address, 428
AK_read_constraint_between, 428

AK_set_constraint_between, 429
bitmap.c
AK_add_to_bitmap_index, 216
AK_bitmap_test, 217
AK_create_Index, 217
AK_create_Index_Table, 218
AK_delete_bitmap_index, 219
AK_get_Attribute, 220
AK_get_attribute, 219
AK_If_ExistOp, 220
AK_print_Att_Test, 220
AK_print_Header_Test, 221
AK_update, 221
bitmap.h
AK_add_to_bitmap_index, 223
AK_bitmap_test, 224
AK_create_Index, 224
AK_create_Index_Table, 225
AK_delete_bitmap_index, 226
AK_get_Attribute, 227
AK_get_attribute, 226
AK_If_ExistOp, 227
AK_print_Att_Test, 227
AK_print_Header_Test, 228
AK_update, 228
AK_write_block, 229
blobs.c
AK_check_folder_blobs, 180
AK_concat, 180
AK_folder_exists, 181
AK_GUID, 181
AK_lo_export, 181
AK_lo_import, 182
AK_lo_test, 182
AK_lo_unlink, 182
AK_mkdir, 183
AK_split_path_file, 183
blobs.h
AK_check_folder_blobs, 185
AK_concat, 185
AK_folder_exists, 185
AK_GUID, 185
AK_lo_export, 186
AK_lo_import, 186
AK_lo_test, 186
AK_lo_unlink, 187
AK_mkdir, 187
AK_split_path_file, 187
blocktable, 25
btree.c
AK_btree_create, 230
AK_btree_search_delete, 231
btree.h
AK_btree_create, 232
AK_btree_search_delete, 232
btree_node, 25
bucket_elem, 26

CHAR_IN_LINE
dbman.h, 163
check_constraint.c
AK_check_constraint, 430
AK_check_constraint_test, 431
AK_set_check_constraint, 431
condition_passed, 432
check_constraint.h
AK_check_constraint, 433
AK_check_constraint_test, 433
AK_set_check_constraint, 433
condition_passed, 434
command.c
AK_command, 421
AK_test_command, 421
command.h
AK_command, 422
AK_test_command, 422
condition_passed
check_constraint.c, 432
check_constraint.h, 434
constants.h
AK_CONSTRAINTS_DEFAULT, 68
AK_CONSTRAINTS_FOREIGN_KEY, 68
AK_CONSTRAINTS_INDEX, 69
AK_CONSTRAINTS_PRIMARY_KEY, 69
constraint_names.c
AK_check_constraint_name, 435
AK_constraint_names_test, 435
constraint_names.h
AK_check_constraint_name, 436
AK_constraint_names_test, 437
cost_eval_t, 26
create_header_test
test.c, 135
test.h, 139

db
dbman.h, 179
db_file_size
dbman.h, 179
dbman.c
AK_allocate_block_activity_modes, 144
AK_allocate_blocks, 144
AK_allocationtable_dump, 144
AK_blocktable_dump, 145
AK_blocktable_flush, 145
AK_blocktable_get, 145
AK_copy_header, 146
AK_create_header, 146
AK_delete_block, 147
AK_delete_extent, 147
AK_delete_segment, 148
AK_get_allocation_set, 148
AK_get_extent, 149
AK_increase_extent, 149
AK_init_allocation_table, 150
AK_init_block, 150
AK_init_db_file, 151
AK_init_disk_manager, 151

AK_init_system_catalog, 152
AK_init_system_tables_catalog, 152
AK_insert_entry, 153
AK_memset_int, 154
AK_new_extent, 154
AK_new_segment, 155
AK_print_block, 156
AK_read_block, 156
AK_read_block_for_testing, 156
AK_register_system_tables, 157
AK_thread_safe_block_access_test, 158
AK_write_block, 158
AK_write_block_for_testing, 159
fsize, 159
dbman.h
AK_allocate_blocks, 164
AK_allocation_set_mode, 163
AK_ALLOCATION_TABLE_SIZE, 163
AK_allocationbit, 179
AK_allocationtable_dump, 164
AK_blocktable_dump, 165
AK_blocktable_flush, 165
AK_blocktable_get, 165
AK_copy_header, 165
AK_create_header, 166
AK_delete_block, 167
AK_delete_extent, 167
AK_delete_segment, 168
AK_get_allocation_set, 168
AK_get_extent, 169
AK_increase_extent, 169
AK_init_allocation_table, 170
AK_init_block, 170
AK_init_db_file, 170
AK_init_disk_manager, 171
AK_init_system_catalog, 171
AK_init_system_tables_catalog, 171
AK_insert_entry, 173
AK_memset_int, 173
AK_new_extent, 174
AK_new_segment, 175
AK_print_block, 175
AK_read_block, 175
AK_read_block_for_testing, 176
AK_register_system_tables, 176
AK_thread_safe_block_access_test, 177
AK_write_block, 178
AK_write_block_for_testing, 178
CHAR_IN_LINE, 163
db, 179
db_file_size, 179
fsize, 178
MAX_BLOCK_INIT_NUM, 163
debug.c
AK_dbg_messg, 70
debug.h
AK_dbg_messg, 71
DEBUG_ALL, 71
DEBUG_ALL
debug.h, 71
DEBUG_LEVEL, 27
DEBUG_TYPE, 27
DICT_INVALID_KEY
dictionary.c, 73
dictionary
dictionary.h, 77
dictionary.c
DICT_INVALID_KEY, 73
dictionary_del, 73
dictionary_dump, 74
dictionary_get, 74
dictionary_hash, 74
dictionary_new, 75
dictionary_set, 75
dictionary_unset, 76
DICTMINSZ, 73
MAXVALSZ, 73
dictionary.h
dictionary, 77
dictionary_del, 77
dictionary_dump, 78
dictionary_get, 78
dictionary_hash, 79
dictionary_new, 79
dictionary_set, 79
dictionary_unset, 80
dictionary_del
dictionary.c, 73
dictionary.h, 77
dictionary_dump
dictionary.c, 74
dictionary.h, 78
dictionary_get
dictionary.c, 74
dictionary.h, 78
dictionary_hash
dictionary.c, 74
dictionary.h, 79
dictionary_new
dictionary.c, 75
dictionary.h, 79
dictionary_set
dictionary.c, 75
dictionary.h, 79
dictionary_unset
dictionary.c, 76
dictionary.h, 80
DICTMINSZ
dictionary.c, 73
difference.c
AK_difference, 376
AK_op_difference_test, 377
difference.h
AK_difference, 377
AK_op_difference_test, 378
dm/dbman.c, 142

dm/dbman.h, 159
drop.c
AK_drop, 464
AK_drop_help_function, 465
AK_drop_test, 465
AK_if_exist, 465
system_catalog, 466
drop.h
AK_drop, 467
AK_drop_test, 467
AK_if_exist, 468
drop_arguments, 28

expression_check.c
AK_check_arithmetic_statement, 379
AK_check_if_row_satisfies_expression, 380
AK_check_regex_expression, 380
AK_check_regex_operator_expression, 381
AK_replace_wild_card, 381
expression_check.h
AK_check_arithmetic_statement, 382
AK_check_if_row_satisfies_expression, 383
AK_check_regex_expression, 384
AK_check_regex_operator_expression, 384

file/blobs.c, 179
file/blobs.h, 184
file/fileio.c, 188
file/fileio.h, 194
file/files.c, 200
file/files.h, 202
file/filesearch.c, 204
file/filesearch.h, 206
file/filesort.h, 209
file/id.c, 212
file/id.h, 214
file/idx/bitmap.c, 215
file/idx/bitmap.h, 222
file/idx/btree.c, 230
file/idx/btree.h, 231
file/idx/hash.c, 233
file/idx/hash.h, 239
file/idx/index.c, 246
file/idx/index.h, 254
file/sequence.c, 262
file/sequence.h, 267
file/table.c, 272
file/table.h, 285
file/test.c, 134
file/test.h, 138
fileio.c
AK_delete_row, 189
AK_delete_row_by_id, 189
AK_delete_row_from_block, 189
AK_delete_update_segment, 190
AK_Insert_New_Element, 190
AK_Insert_New_Element_For_Update, 191
AK_insert_row, 192
AK_insert_row_to_block, 192
AK_Update_Existing_Element, 193
AK_update_row, 193
AK_update_row_from_block, 194
fileio.h
AK_delete_row, 195
AK_delete_row_by_id, 195
AK_delete_row_from_block, 196
AK_delete_update_segment, 196
AK_Insert_New_Element, 197
AK_Insert_New_Element_For_Update, 197
AK_insert_row, 198
AK_insert_row_to_block, 199
AK_update_row, 199
AK_update_row_from_block, 199
files.c
AK_files_test, 201
AK_initialize_new_index_segment, 201
AK_initialize_new_segment, 201
files.h
AK_files_test, 203
AK_initialize_new_index_segment, 203
AK_initialize_new_segment, 203
filesearch.c
AK_deallocate_search_result, 205
AK_filesearch_test, 205
AK_search_unsorted, 205
filesearch.h
AK_deallocate_search_result, 207
AK_filesearch_test, 208
AK_search_unsorted, 208
filesort.h
AK_block_sort, 209
AK_get_header_number, 210
AK_get_num_of_tuples, 210
AK_get_total_headers, 211
AK_reset_block, 211
AK_sort_segment, 212
fsize
dbman.c, 159
dbman.h, 178
function.c
AK_check_function_arguments, 469
AK_check_function_arguments_type, 469
AK_function_add, 470
AK_function_arguments_add, 470
AK_function_arguments_remove_by_obj_id, 471
AK_function_change_return_type, 471
AK_function_remove_by_name, 472
AK_function_remove_by_obj_id, 472
AK_function_rename, 473
AK_function_test, 473
AK_get_function_obj_id, 474
function.h
AK_check_function_arguments, 475
AK_check_function_arguments_type, 476
AK_function_add, 476
AK_function_arguments_add, 477
AK_function_arguments_remove_by_obj_id, 477

AK_function_change_return_type, 478
AK_function_remove_by_name, 478
AK_function_remove_by_obj_id, 479
AK_function_rename, 479
AK_function_test, 480
AK_get_function_obj_id, 480

get_column_test
test.c, 135
test.h, 139
get_row_attr_data
table.c, 285
table.h, 298
get_row_test
test.c, 136
test.h, 140
grandfailure
recovery.c, 363

handle_transaction_notify
transaction.c, 540
transaction.h, 557
hash
dictionary, 11
hash.c
AK_change_hash_info, 234
AK_create_hash_index, 234
AK_delete_in_hash_index, 235
AK_elem_hash_value, 235
AK_find_delete_in_hash_index, 235
AK_find_in_hash_index, 236
AK_get_hash_info, 236
AK_get_nth_main_bucket_add, 237
AK_hash_test, 237
AK_insert_bucket_to_block, 238
AK_insert_in_hash_index, 238
AK_update_bucket_in_block, 239
hash.h
AK_change_hash_info, 240
AK_create_hash_index, 241
AK_delete_in_hash_index, 241
AK_elem_hash_value, 242
AK_find_delete_in_hash_index, 242
AK_find_in_hash_index, 243
AK_get_hash_info, 243
AK_get_nth_main_bucket_add, 244
AK_hash_test, 244
AK_insert_bucket_to_block, 244
AK_insert_in_hash_index, 245
AK_update_bucket_in_block, 245
hash_bucket, 28
hash_info, 29

id.c
AK_get_id, 213
AK_get_table_id, 213
AK_id_test, 213
id.h
AK_get_id, 214
AK_id_test, 215
index.c
AK_Delete_All_elementsAd, 247
AK_Delete_elementAd, 247
AK_Get_First_elementAd, 248
AK_get_index_header, 248
AK_get_index_num_records, 249
AK_get_index_tuple, 249
AK_Get_Last_elementAd, 250
AK_Get_Next_elementAd, 250
AK_Get_Position_Of_elementAd, 251
AK_Get_Previous_elementAd, 251
AK_index_table_exist, 252
AK_index_test, 252
AK_InitializelistAd, 252
AK_Insert_NewelementAd, 253
AK_num_index_attr, 253
AK_print_index_table, 254
index.h
AK_Delete_All_elementsAd, 256
AK_Delete_elementAd, 256
AK_Get_First_elementAd, 256
AK_get_index_num_records, 257
AK_get_index_tuple, 257
AK_Get_Last_elementAd, 258
AK_Get_Next_elementAd, 258
AK_Get_Position_Of_elementAd, 259
AK_Get_Previous_elementAd, 259
AK_index_table_exist, 260
AK_index_test, 260
AK_InitializelistAd, 260
AK_Insert_NewelementAd, 261
AK_num_index_attr, 261
AK_print_index_table, 262
iniparser.c
_line_status_, 82
iniparser_AK_freedict, 82
iniparser_dump, 83
iniparser_dump_ini, 83
iniparser_dumpsection_ini, 84
iniparser_find_entry, 84
iniparser_getboolean, 84
iniparser_getdouble, 85
iniparser_getint, 86
iniparser_getnsec, 86
iniparser_getseckeys, 87
iniparser_getsecname, 87
iniparser_getsecnkeys, 88
iniparser_getstring, 88
iniparser_load, 88
iniparser_set, 89
iniparser_unset, 89
line_status, 82
iniparser.h
iniparser_AK_freedict, 91
iniparser_dump, 91
iniparser_dump_ini, 92
iniparser_dumpsection_ini, 92

iniparser_find_entry, 93
iniparser_getboolean, 93
iniparser_getdouble, 94
iniparser_getint, 94
iniparser_getnsec, 95
iniparser_getseckeys, 96
iniparser_getsecname, 96
iniparser_getsecnkeys, 97
iniparser_getstring, 97
iniparser_load, 97
iniparser_set, 98
iniparser_unset, 98
iniparser_AK_freedict
iniparser.c, 82
iniparser.h, 91
iniparser_dump
iniparser.c, 83
iniparser.h, 91
iniparser_dump_ini
iniparser.c, 83
iniparser.h, 92
iniparser_dumpsection_ini
iniparser.c, 84
iniparser.h, 92
iniparser_find_entry
iniparser.c, 84
iniparser.h, 93
iniparser_getboolean
iniparser.c, 84
iniparser.h, 93
iniparser_getdouble
iniparser.c, 85
iniparser.h, 94
iniparser_getint
iniparser.c, 86
iniparser.h, 94
iniparser_getnsec
iniparser.c, 86
iniparser.h, 95
iniparser_getseckeys
iniparser.c, 87
iniparser.h, 96
iniparser_getsecname
iniparser.c, 87
iniparser.h, 96
iniparser_getsecnkeys
iniparser.c, 88
iniparser.h, 97
iniparser_getstring
iniparser.c, 88
iniparser.h, 97
iniparser_load
iniparser.c, 88
iniparser.h, 97
iniparser_set
iniparser.c, 89
iniparser.h, 98
iniparser_unset
iniparser.c, 89
iniparser.h, 98
insert.h
AK_get_insert_header, 481
AK_insert, 482
insert_data_test
test.c, 136
test.h, 140
intersect.c
AK_intersect, 385
AK_op_intersect_test, 386
intersect.h
AK_intersect, 387
AK_op_intersect_test, 387
intersect_attr, 29

key
dictionary, 11

line_status
iniparser.c, 82
list_node, 30
list_structure_ad, 31
list_structure_add, 31

main_bucket, 31
MAX_BLOCK_INIT_NUM
dbman.h, 163
MAXVALSZ
dictionary.c, 73
memoman.c
AK_cache_AK_malloc, 299
AK_cache_block, 300
AK_cache_result, 300
AK_find_AK_free_space, 301
AK_find_available_result_block, 301
AK_flush_cache, 301
AK_generate_result_id, 302
AK_get_block, 302
AK_get_index_addresses, 303
AK_get_index_segment_addresses, 303
AK_get_segment_addresses, 304
AK_get_segment_addresses_internal, 304
AK_get_system_table_address, 305
AK_get_table_addresses, 305
AK_init_new_extent, 305
AK_mem_block_modify, 306
AK_memoman_init, 306
AK_query_mem_AK_free, 306
AK_query_mem_AK_malloc, 307
AK_redo_log_AK_malloc, 307
AK_refresh_cache, 307
AK_release_oldest_cache_block, 308
memoman.h
AK_cache_AK_malloc, 310
AK_cache_block, 310
AK_cache_result, 311
AK_find_AK_free_space, 311
AK_find_available_result_block, 312

AK_flush_cache, 312
AK_generate_result_id, 312
AK_get_block, 313
AK_get_index_addresses, 313
AK_get_index_segment_addresses, 314
AK_get_segment_addresses, 314
AK_get_segment_addresses_internal, 315
AK_get_table_addresses, 315
AK_init_new_extent, 316
AK_mem_block_modify, 316
AK_memoman_init, 317
AK_query_mem_AK_free, 317
AK_query_mem_AK_malloc, 317
AK_redo_log_AK_malloc, 318
AK_refresh_cache, 318
AK_release_oldest_cache_block, 318
memoryAddresses, 32
mempro.c
AK_calloc, 100
AK_check_for_writes, 101
AK_debmod_calloc, 101
AK_debmod_d, 102
AK_debmod_die, 102
AK_debmod_dv, 102
AK_debmod_enter_critical_sec, 103
AK_debmod_free, 103
AK_debmod_fstack_pop, 104
AK_debmod_fstack_push, 104
AK_debmod_func_add, 105
AK_debmod_func_get_name, 105
AK_debmod_func_id, 106
AK_debmod_function_current, 106
AK_debmod_function_epilogue, 107
AK_debmod_function_prologue, 107
AK_debmod_init, 108
AK_debmod_leave_critical_sec, 108
AK_debmod_log_memory_alloc, 108
AK_debmod_print_function_use, 109
AK_fread, 109
AK_free, 110
AK_fwrite, 110
AK_malloc, 111
AK_mempro_test, 111
AK_print_active_functions, 111
AK_print_function_use, 111
AK_print_function_uses, 112
AK_realloc, 112
AK_write_protect, 113
AK_write_unprotect, 113
mempro.h
AK_calloc, 116
AK_check_for_writes, 116
AK_debmod_calloc, 117
AK_debmod_d, 117
AK_debmod_die, 118
AK_debmod_dv, 118
AK_debmod_enter_critical_sec, 119
AK_debmod_free, 119
AK_debmod_fstack_pop, 119
AK_debmod_fstack_push, 120
AK_debmod_func_add, 120
AK_debmod_func_get_name, 121
AK_debmod_func_id, 121
AK_debmod_function_current, 122
AK_debmod_function_epilogue, 122
AK_debmod_function_prologue, 123
AK_debmod_init, 123
AK_debmod_leave_critical_sec, 123
AK_debmod_log_memory_alloc, 124
AK_debmod_print_function_use, 124
AK_free, 125
AK_malloc, 125
AK_mempro_test, 126
AK_print_active_functions, 126
AK_print_function_use, 126
AK_print_function_uses, 127
AK_realloc, 127
AK_write_protect, 127
AK_write_unprotect, 128
mm/memoman.c, 298
mm/memoman.h, 308

nat_join.c
AK_copy_blocks_join, 388
AK_create_join_block_header, 389
AK_join, 389
AK_merge_block_join, 390
AK_op_join_test, 390
nat_join.h
AK_copy_blocks_join, 391
AK_create_join_block_header, 392
AK_join, 392
AK_merge_block_join, 393
AK_op_join_test, 393
nnull.c
AK_check_constraint_not_null, 438
AK_delete_constraint_not_null, 438
AK_nnull_constraint_test, 439
AK_read_constraint_not_null, 439
AK_set_constraint_not_null, 440
nnull.h
AK_check_constraint_not_null, 441
AK_delete_constraint_not_null, 441
AK_nnull_constraint_test, 442
AK_read_constraint_not_null, 442
AK_set_constraint_not_null, 443
NoticeType
transaction.h, 543

Observable, 32
observable.c
AK_init_observable, 129
AK_init_observer, 130
AK_observable_test, 130
observable.h
AK_init_observable, 131
AK_init_observer, 132

AK_observable_test, 132
observable_transaction, 33
observable_transaction_struct, 33
Observer, 34
observer_lock, 34
opti/query_optimization.c, 319
opti/query_optimization.h, 321
opti/rel_eq_assoc.c, 324
opti/rel_eq_assoc.h, 326
opti/rel_eq_comut.c, 329
opti/rel_eq_comut.h, 331
opti/rel_eq_projection.c, 333
opti/rel_eq_projection.h, 339
opti/rel_eq_selection.c, 345
opti/rel_eq_selection.h, 350

privileges.c
AK_add_user_to_group, 483
AK_check_group_privilege, 484
AK_check_privilege, 484
AK_check_user_privilege, 485
AK_grant_privilege_group, 485
AK_grant_privilege_user, 486
AK_group_add, 486
AK_group_get_id, 487
AK_group_remove_by_name, 487
AK_group_rename, 487
AK_privileges_test, 488
AK_remove_all_users_from_group, 488
AK_remove_user_from_all_groups, 489
AK_revoke_all_privileges_group, 489
AK_revoke_all_privileges_user, 490
AK_revoke_privilege_group, 490
AK_revoke_privilege_user, 491
AK_user_add, 491
AK_user_check_pass, 492
AK_user_get_id, 492
AK_user_remove_by_name, 492
AK_user_rename, 493
privileges.h
AK_add_user_to_group, 495
AK_check_group_privilege, 495
AK_check_privilege, 496
AK_check_user_privilege, 496
AK_grant_privilege_group, 497
AK_grant_privilege_user, 497
AK_group_add, 498
AK_group_get_id, 498
AK_group_remove_by_name, 499
AK_group_rename, 499
AK_privileges_test, 499
AK_remove_all_users_from_group, 500
AK_remove_user_from_all_groups, 500
AK_revoke_all_privileges_group, 501
AK_revoke_all_privileges_user, 501
AK_revoke_privilege_group, 502
AK_revoke_privilege_user, 502
AK_user_add, 503
AK_user_check_pass, 504
AK_user_get_id, 504
AK_user_rename, 505
product.c
AK_op_product_test, 394
AK_product, 394
AK_product_procedure, 395
product.h
AK_op_product_test, 396
AK_product, 396
AK_product_procedure, 397
projection.c
AK_copy_block_projection, 398
AK_create_block_header, 398
AK_create_header_name, 399
AK_determine_header_type, 400
AK_get_operator, 400
AK_op_projection_test, 400
AK_perform_operation, 401
AK_projection, 401
AK_remove_substring, 402
projection.h
AK_copy_block_projection, 403
AK_create_block_header, 404
AK_create_header_name, 405
AK_determine_header_type, 405
AK_get_operator, 406
AK_op_projection_test, 406
AK_perform_operation, 407
AK_projection, 407
AK_remove_substring, 408

query_optimization.c
AK_execute_rel_eq, 319
AK_print_optimized_query, 320
AK_query_optimization, 320
AK_query_optimization_test, 321
query_optimization.h
AK_execute_rel_eq, 322
AK_print_optimized_query, 323
AK_query_optimization, 323
AK_query_optimization_test, 324

rec/archive_log.h, 357
rec/recovery.c, 359
rec/redo_log.c, 364
recovery.c
AK_load_chosen_log, 360
AK_load_latest_log, 360
AK_recover_archive_log, 361
AK_recover_operation, 361
AK_recovery_insert_row, 362
AK_recovery_test, 362
AK_recovery_tokenize, 362
grandfailure, 363
recovery_insert_row, 363
recovery_insert_row
recovery.c, 363
redo_log.c
AK_add_to_redolog, 364

AK_add_to_redolog_select, 364
AK_check_attributes, 365
AK_check_redo_log_select, 365
AK_printout_redolog, 365
REF_TYPE_NO_ACTION
reference.h, 450
reference.c
AK_add_reference, 444
AK_get_reference, 445
AK_reference_check_attribute, 445
AK_reference_check_entry, 446
AK_reference_check_if_update_needed, 446
AK_reference_check_restricion, 446
AK_reference_test, 447
AK_reference_update, 447
reference.h
AK_add_reference, 450
AK_delete_row, 450
AK_get_reference, 451
AK_initialize_new_segment, 451
AK_Insert_New_Element, 452
AK_Insert_New_Element_For_Update, 452
AK_insert_row, 453
AK_reference_check_attribute, 454
AK_reference_check_entry, 454
AK_reference_check_if_update_needed, 455
AK_reference_check_restricion, 455
AK_reference_test, 456
AK_reference_update, 456
AK_selection, 456
AK_Update_Existing_Element, 457
AK_update_row, 457
REF_TYPE_NO_ACTION, 450
rel/aggregation.c, 366
rel/aggregation.h, 371
rel/difference.c, 376
rel/difference.h, 377
rel/expression_check.c, 378
rel/expression_check.h, 381
rel/intersect.c, 385
rel/intersect.h, 386
rel/nat_join.c, 387
rel/nat_join.h, 391
rel/product.c, 394
rel/product.h, 395
rel/projection.c, 397
rel/projection.h, 402
rel/selection.c, 408
rel/selection.h, 410
rel/theta_join.c, 412
rel/theta_join.h, 414
rel/union.c, 417
rel/union.h, 419
rel_eq_assoc.c
AK_compare, 325
AK_print_rel_eq_assoc, 325
AK_rel_eq_assoc, 326
AK_rel_eq_assoc_test, 326
rel_eq_assoc.h
AK_compare, 327
AK_print_rel_eq_assoc, 328
AK_rel_eq_assoc, 328
AK_rel_eq_assoc_test, 328
rel_eq_comut.c
AK_print_rel_eq_comut, 329
AK_rel_eq_commute_with_theta_join, 330
AK_rel_eq_comut, 330
AK_rel_eq_comut_test, 331
rel_eq_comut.h
AK_print_rel_eq_comut, 332
AK_rel_eq_commute_with_theta_join, 332
AK_rel_eq_comut, 333
AK_rel_eq_comut_test, 333
rel_eq_projection.c
AK_print_rel_eq_projection, 334
AK_rel_eq_can_commute, 335
AK_rel_eq_collect_cond_attributes, 335
AK_rel_eq_get_attributes, 336
AK_rel_eq_is_subset, 336
AK_rel_eq_projection, 337
AK_rel_eq_projection_attributes, 338
AK_rel_eq_projection_test, 338
AK_rel_eq_remove_duplicates, 339
rel_eq_projection.h
AK_print_rel_eq_projection, 340
AK_rel_eq_can_commute, 340
AK_rel_eq_collect_cond_attributes, 341
AK_rel_eq_get_attributes, 341
AK_rel_eq_is_subset, 342
AK_rel_eq_projection, 343
AK_rel_eq_projection_attributes, 344
AK_rel_eq_projection_test, 344
AK_rel_eq_remove_duplicates, 345
rel_eq_selection.c
AK_print_rel_eq_selection, 346
AK_rel_eq_cond_attributes, 346
AK_rel_eq_get_atrributes_char, 347
AK_rel_eq_is_attr_subset, 347
AK_rel_eq_selection, 348
AK_rel_eq_selection_test, 348
AK_rel_eq_share_attributes, 349
AK_rel_eq_split_condition, 349
rel_eq_selection.h
AK_print_rel_eq_selection, 351
AK_rel_eq_cond_attributes, 351
AK_rel_eq_get_atrributes_char, 352
AK_rel_eq_is_attr_subset, 354
AK_rel_eq_selection, 355
AK_rel_eq_selection_test, 355
AK_rel_eq_share_attributes, 355
AK_rel_eq_split_condition, 356
root_info, 35

search_params, 35
search_result, 36
select.c
AK_select, 506

AK_select_test, 506
select.h
AK_select, 507
AK_select_test, 508
selection.c
AK_op_selection_test, 409
AK_op_selection_test_pattern, 409
AK_selection, 409
AK_selection_op_rename, 410
selection.h
AK_op_selection_test, 411
AK_op_selection_test_pattern, 411
AK_selection, 411
selection_test
test.c, 137
test.h, 141
sequence.c
AK_sequence_add, 263
AK_sequence_current_value, 264
AK_sequence_get_id, 264
AK_sequence_modify, 264
AK_sequence_next_value, 265
AK_sequence_remove, 266
AK_sequence_rename, 266
AK_sequence_test, 266
sequence.h
AK_sequence_add, 268
AK_sequence_current_value, 268
AK_sequence_get_id, 269
AK_sequence_modify, 269
AK_sequence_next_value, 270
AK_sequence_remove, 270
AK_sequence_rename, 270
AK_sequence_test, 271
size
dictionary, 12
sql/command.c, 420
sql/command.h, 421
sql/cs/between.c, 423
sql/cs/between.h, 426
sql/cs/check_constraint.c, 430
sql/cs/check_constraint.h, 432
sql/cs/constraint_names.c, 435
sql/cs/constraint_names.h, 436
sql/cs/nnull.c, 437
sql/cs/nnull.h, 440
sql/cs/reference.c, 443
sql/cs/reference.h, 448
sql/cs/unique.c, 458
sql/cs/unique.h, 461
sql/drop.c, 463
sql/drop.h, 466
sql/function.c, 468
sql/function.h, 474
sql/insert.h, 480
sql/privileges.c, 482
sql/privileges.h, 493
sql/select.c, 505
sql/select.h, 507
sql/trigger.c, 508
sql/trigger.h, 513
sql/view.c, 518
Stack, 37
struct_add, 37
Succesor, 38
system_catalog
drop.c, 466

table.c
AK_check_tables_scheme, 273
AK_create_table, 273
AK_get_attr_index, 274
AK_get_attr_name, 274
AK_get_column, 275
AK_get_header, 275
AK_get_num_records, 277
AK_get_row, 277
AK_get_table_obj_id, 278
AK_get_tuple, 278
AK_num_attr, 279
AK_op_rename_test, 279
AK_print_row, 279
AK_print_row_spacer, 280
AK_print_row_spacer_to_file, 280
AK_print_row_to_file, 281
AK_print_table, 281
AK_print_table_to_file, 282
AK_rename, 282
AK_table_empty, 283
AK_table_exist, 283
AK_table_test, 283
AK_temp_create_table, 284
AK_tuple_to_string, 284
get_row_attr_data, 285
table.h
AK_check_tables_scheme, 287
AK_create_table, 287
AK_get_attr_index, 288
AK_get_attr_name, 288
AK_get_column, 289
AK_get_header, 289
AK_get_num_records, 290
AK_get_row, 290
AK_get_table_obj_id, 291
AK_get_tuple, 291
AK_num_attr, 292
AK_op_rename_test, 292
AK_print_row, 293
AK_print_row_spacer, 293
AK_print_row_spacer_to_file, 294
AK_print_row_to_file, 294
AK_print_table, 295
AK_print_table_to_file, 295
AK_rename, 295
AK_table_empty, 296
AK_table_test, 296
AK_temp_create_table, 297

AK_tuple_to_string, 297
get_row_attr_data, 298
table_addresses, 38
test.c
AK_create_test_tables, 134
AK_get_table_atribute_types, 134
create_header_test, 135
get_column_test, 135
get_row_test, 136
insert_data_test, 136
selection_test, 137
TEST_output_results, 133
TEST_result, 133
test.h
AK_create_test_tables, 138
AK_get_table_atribute_types, 138
create_header_test, 139
get_column_test, 139
get_row_test, 140
insert_data_test, 140
selection_test, 141
TEST_output_results
test.c, 133
TEST_result
test.c, 133
testMode
auxiliary.h, 64
TestResult, 39
theta_join.c
AK_check_constraints, 412
AK_create_theta_join_header, 413
AK_op_theta_join_test, 413
AK_theta_join, 414
theta_join.h
AK_check_constraints, 415
AK_create_theta_join_header, 416
AK_op_theta_join_test, 416
AK_theta_join, 416
threadContainer, 39
tools/comments.py, 524
tools/getFiles.sh, 524
tools/parseC.sh, 524
tools/parsePy.sh, 525
tools/updateVersion.sh, 525
trans/transaction.c, 525
trans/transaction.h, 540
transaction.c
AK_acquire_lock, 527
AK_add_hash_entry_list, 528
AK_add_lock, 528
AK_all_transactions_finished, 529
AK_create_lock, 529
AK_create_new_transaction_thread, 529
AK_delete_hash_entry_list, 530
AK_delete_lock_entry_list, 530
AK_execute_commands, 531
AK_execute_transaction, 531
AK_get_memory_blocks, 532
AK_handle_observable_transaction_action, 532
AK_init_observable_transaction, 532
AK_init_observer_lock, 533
AK_isLock_waiting, 533
AK_lock_released, 534
AK_memory_block_hash, 534
AK_on_all_transactions_end, 535
AK_on_lock_release, 535
AK_on_observable_notify, 535
AK_on_transaction_end, 536
AK_release_locks, 536
AK_remove_transaction_thread, 536
AK_search_empty_link_for_hook, 537
AK_search_existing_link_for_hook, 537
AK_search_lock_entry_list_by_key, 538
AK_transaction_finished, 538
AK_transaction_manager, 538
AK_transaction_register_observer, 539
AK_transaction_unregister_observer, 539
handle_transaction_notify, 540
transaction.h
AK_acquire_lock, 543
AK_add_hash_entry_list, 544
AK_add_lock, 545
AK_all_transactions_finished, 545
AK_create_lock, 545
AK_create_new_transaction_thread, 546
AK_delete_hash_entry_list, 546
AK_delete_lock_entry_list, 547
AK_execute_commands, 547
AK_execute_transaction, 548
AK_get_memory_blocks, 548
AK_handle_observable_transaction_action, 549
AK_init_observable_transaction, 549
AK_init_observer_lock, 549
AK_isLock_waiting, 550
AK_lock_released, 550
AK_memory_block_hash, 551
AK_on_all_transactions_end, 551
AK_on_lock_release, 552
AK_on_observable_notify, 552
AK_on_transaction_end, 552
AK_release_locks, 553
AK_remove_transaction_thread, 553
AK_search_empty_link_for_hook, 554
AK_search_existing_link_for_hook, 554
AK_search_lock_entry_list_by_key, 554
AK_transaction_finished, 555
AK_transaction_manager, 555
AK_transaction_register_observer, 556
AK_transaction_unregister_observer, 556
handle_transaction_notify, 557
NoticeType, 543
transaction_list_elem, 40
transaction_list_head, 41
transaction_locks_list_elem, 41
transactionData, 42
trigger.c

AK_trigger_add, 509
AK_trigger_edit, 509
AK_trigger_get_conditions, 510
AK_trigger_get_id, 510
AK_trigger_remove_by_name, 511
AK_trigger_remove_by_obj_id, 511
AK_trigger_rename, 512
AK_trigger_save_conditions, 512
AK_trigger_test, 513
trigger.h
AK_trigger_add, 514
AK_trigger_edit, 514
AK_trigger_get_conditions, 515
AK_trigger_get_id, 516
AK_trigger_remove_by_name, 516
AK_trigger_remove_by_obj_id, 517
AK_trigger_rename, 517
AK_trigger_save_conditions, 518
AK_trigger_test, 518
TypeObservable, 42
TypeObserver, 42

union.c
AK_op_union_test, 418
AK_union, 418
union.h
AK_op_union_test, 419
AK_union, 419
unique.c
AK_delete_constraint_unique, 458
AK_read_constraint_unique, 459
AK_set_constraint_unique, 460
AK_unique_test, 460
unique.h
AK_delete_constraint_unique, 461
AK_read_constraint_unique, 462
AK_set_constraint_unique, 462
AK_unique_test, 463

val
dictionary, 12
Vertex, 43
view.c
AK_check_view_name, 519
AK_get_rel_exp, 520
AK_get_view_obj_id, 520
AK_get_view_query, 520
AK_test_get_view_data, 521
AK_view_add, 521
AK_view_change_query, 522
AK_view_remove_by_name, 522
AK_view_remove_by_obj_id, 523
AK_view_rename, 523
AK_view_test, 523