ECE297 Storage Server

0.2

Generated by Doxygen 1.8.1.2

Sat Apr 12 2014 12:34:29

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 arguements Struct Reference

Encapsulate the value associated with a key in a table.

`#include <utils.h>`

Public Attributes

- int sock_

  Communication port value.
- struct sockaddr_in clientaddr_

  The client address information.
- socklen_t clientaddrlen_

  Socket length.

3.1.1 Detailed Description

Encapsulate the value associated with a key in a table.

Definition at line 110 of file utils.h.

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

- utils.h

3.2 config_params Struct Reference

A struct to store config parameters.

`#include <utils.h>`

Public Attributes

- char server_host [MAX_HOST_LEN]

The hostname of the server.

- int server_port

  The listening port of the server.

- char username [MAX_USERNAME_LEN]

  The storage server's username.

- char password [MAX_ENC_PASSWORD_LEN]

  The storage server's encrypted password.

- char **mytables** [MAX_TABLES][MAX_TABLE_LEN]
- char **mycolumns** [MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- int **tablecount**
- int **numcolumnspertable** [MAX_TABLES]
- char **column_types** [MAX_TABLES][MAX_COLUMNS_PER_TABLE][10]
- int **storage_policy**
- char **data_directory** [MAX_PATH_LEN]
- int **concurrency**
- pthread_mutex_t **lock**

3.2.1 Detailed Description

A struct to store config parameters.

Definition at line 53 of file utils.h.

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

- utils.h

3.3 server_record Struct Reference

Encapsulate the value associated with a key in a table.

`#include <utils.h>`

Public Attributes

- char value [MAX_VALUE_LEN]

  This is where the actual value is stored.

- char key [MAX_KEY_LEN]

  This is where the key is stored.

- unsigned long metadata

  A place to put any extra data.

- pthread_mutex_t lock

  mutex variable for locking

3.3.1 Detailed Description

Encapsulate the value associated with a key in a table.

The metadata will be used later.

Definition at line 92 of file utils.h.

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

- utils.h

3.4 storage_record Struct Reference

Encapsulate the value associated with a key in a table.

`#include <storage.h>`

Public Attributes

- char value [MAX_VALUE_LEN]
  This is where the actual value is stored.
- uintptr_t metadata [8]
  A place to put any extra data.

3.4.1 Detailed Description

Encapsulate the value associated with a key in a table.

The metadata will be used later.

Definition at line 54 of file storage.h.

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

- storage.h

3.5 yy_buffer_state Struct Reference

Public Attributes

- FILE * **yy_input_file**
- char * **yy_ch_buf**
- char * **yy_buf_pos**
- yy_size_t **yy_buf_size**
- int **yy_n_chars**
- int **yy_is_our_buffer**
- int **yy_is_interactive**
- int **yy_at_bol**
- int yy_bs_lineno
- int yy_bs_column
- int **yy_fill_buffer**
- int **yy_buffer_status**

3.5.1 Detailed Description

Definition at line 197 of file lex.yy.c.

3.5.2 Member Data Documentation

3.5.2.1 int yy_buffer_state::yy_bs_column

The column count.

Definition at line 234 of file lex.yy.c.

3.5.2.2 int yy_buffer_state::yy_bs_lineno

The line count.

Definition at line 233 of file lex.yy.c.

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

- lex.yy.c

3.6 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t **yy_verify**
- flex_int32_t **yy_nxt**

3.6.1 Detailed Description

Definition at line 375 of file lex.yy.c.

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

- lex.yy.c

3.7 yyalloc Union Reference

Public Attributes

- yytype_int16 **yyss_alloc**
- YYSTYPE **yyvs_alloc**

3.7.1 Detailed Description

Definition at line 347 of file parser.tab.c.

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

- parser.tab.c

3.8 YYSTYPE Union Reference

Public Attributes

- char * **stringVal**
- char * **passwordVal**
- char * **hostVal**
- char * **dataVal**
- int **intVal**

3.8.1 Detailed Description

Definition at line 159 of file parser.tab.c.

The documentation for this union was generated from the following files:

- parser.tab.c
- parser.tab.h

Chapter 4

File Documentation

4.1 encrypt_passwd.c File Reference

This program implements a password encryptor.

```
#include <stdlib.h>
#include <stdio.h>
#include "utils.h"
```

Functions

- void print_usage ()

  Print the usage to stdout.

- int **main** (int argc, char *argv[])

4.1.1 Detailed Description

This program implements a password encryptor.

Definition in file encrypt_passwd.c.

4.2 server.c File Reference

This file implements the storage server.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <assert.h>
#include <signal.h>
#include "utils.h"
#include <time.h>
#include <stdbool.h>
#include <ctype.h>
#include <math.h>
#include <sys/resource.h>
#include <sys/stat.h>
#include <pthread.h>
```

Macros

- #define MAX_LISTENQUEUELEN 20

  The maximum number of queued connections.

Functions

- int key_exist (char key_to_search_for[MAX_KEY_LEN], int first_empty, int table_num)

  Check if key exists in the server.
- char * get_column_type (char column_name[MAX_COLNAME_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10])

  Get the name of the column based on name given to search for.
- void split_query_get_column (char pred[MAX_VALUE_LEN], char ret[MAX_VALUE_LEN])

  Get the column name from a query predicate.
- void split_query_get_value (char pred[MAX_VALUE_LEN], char ret[MAX_VALUE_LEN])

  Get the comparison value from a query predicate.
- char * trim (char *str)

  Remove whitespace from beginning and end of string.
- int has_table (char table_name[MAX_TABLE_LEN])

  Check if table exists in the server.
- int has_column (char col_to_search[MAX_COLNAME_LEN], char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], int num_columns, int table_num)

  Check if table exists in the server.
- int check_column_types (char val_to_set[MAX_VALUE_LEN], int num_columns, int table_num, char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10])

  Check if the column types match the values given.
- int check_mycolumns (char val_to_set[MAX_VALUE_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN])

  Check if the value has the correct matching column names.

- int num_col_val (char val_to_set[MAX_VALUE_LEN], int num_columns)

  Check if the value has the correct number of columns.
- int parse_value (char val_to_set[MAX_VALUE_LEN], int table_num)

  Checks if the value passes all parsing.
- int check_predicates (char predicates[MAX_VALUE_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10], int num_pred)

  Check if all the predicates in the string are formatted correctly.
- int check_mycolumns_pred (char pred_to_set[MAX_VALUE_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], int num_pred)

  Check if the predicate has the correct matching column names.
- int num_of_predicates (char predicates[MAX_VALUE_LEN], int num_columns)

  Check if the predicate string has the correct number of predicates.
- int parse_predicates (char predicates[MAX_VALUE_LEN], int table_num)

  Umbrella function for all predicate parsing.
- void get_command (char key_to_get[MAX_KEY_LEN], char value_to_get[MAX_VALUE_LEN], int first_empty, int table_num)

  Get the specified value based on the key_to_get.
- void **get_command_perm** (char key_to_get[MAX_KEY_LEN], FILE *fileLoadData, char value_to_get[MAX_VALUE_LEN])
- void **get_command_specific_perm** (int lineNumber, FILE *fileLoadData, char value_to_get[MAX_VALUE_LEN])
- char * set_command (char key_to_set[MAX_KEY_LEN], char value_to_set[MAX_VALUE_LEN], int first_empty, int table_num)

  Set the item using key_to_set and value_to_set.
- char * **set_command_perm** (char key_to_set[MAX_KEY_LEN], char value_to_set[MAX_VALUE_LEN], FILE *fileLoadData, FILE *fileWriteData)
- char * update_command (char key_to_update[MAX_KEY_LEN], char value_to_update[MAX_VALUE_LEN], int record_loc, int table_num, unsigned long int meta_data_recieved)

  Set the item using key_to_set and value_to_set.
- int get_column_index (char column_name[MAX_COLNAME_LEN], char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], int table_num)

  Get the index of the column based on the column name given.
- int predicate_true (char predicate[MAX_VALUE_LEN], int table_num, int column_index, int row_index, char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10])

  Check if the value in the row index and column index passes the predicate.
- int **predicate_true_perm** (char predicate[MAX_VALUE_LEN], char lineFromFile[MAX_VALUE_LEN], int table_num, int column_index, int row_index, char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10])
- int predicates_true (char predicates[MAX_VALUE_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10], int row_index)

  Check if the value in the row index and column index passes the predicates.
- int **predicates_true_perm** (char predicates[MAX_VALUE_LEN], char lineFromFile[MAX_VALUE_LEN], int num_columns, int table_num, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10], int row_index)
- int query_command (int sock, char predicates[MAX_VALUE_LEN], int first_empty, int table_num, int num_columns, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10])

  Query the table for matching values.

- int **query_command_perm** (int sock, char predicates[MAX_VALUE_LEN], int table_num, int num_columns, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN], char column_types[MAX_-TABLES][MAX_COLUMNS_PER_TABLE][10], FILE ∗fileToLoad)
- char ∗ delete_command (char key_to_delete[MAX_KEY_LEN], int first_empty, int table_num, int num_columns, char mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN])

  Delete the item in the server based on key_to_delete.
- char ∗ **delete_command_perm** (char key_to_set[MAX_KEY_LEN], FILE ∗fileLoadData, FILE ∗fileWriteData)
- int handle_command (int sock, char cmd[MAX_CMD_LEN], int ∗auth_var)

  Process a command from the client.
- void ∗ **handle_client** (void ∗arg)
- int main (int argc, char ∗argv[])

  Start the storage server.

Variables

- FILE ∗ **fserverOut**
- struct server_record **tables** [MAX_TABLES][MAX_RECORDS_PER_TABLE]
- int **first_empty** [MAX_TABLES]
- struct config_params **params**

4.2.1 Detailed Description

This file implements the storage server. The storage server should be named "server" and should take a single command line argument that refers to the configuration file.

The storage server should be able to communicate with the client library functions declared in storage.h and implemented in storage.c.

Definition in file server.c.

4.2.2 Function Documentation

4.2.2.1 int check_column_types (char *val_to_set[MAX_VALUE_LEN],* int *num_columns,* int *table_num,* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10]*)

Check if the column types match the values given.

Parameters

val_to_set	value to parse
num_columns	number of columns for the table
table_num	index of the table parsing
column_types	types of the columns

Returns

returns true(1) if all column types match, false(0) if it doesn't

Definition at line 260 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by parse_value().

4.2.2.2 int check_mycolumns (char *val_to_set[MAX_VALUE_LEN],* int *num_columns,* int *table_num,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]*)

Check if the value has the correct matching column names.

Parameters

val_to_set	value to parse
num_columns	number of columns for the table
table_num	index of the table parsing
mycolumns	names of the columns

Returns

returns true(1) if all column names match, false(-1) if it doesn't

Definition at line 360 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by parse_value().

4.2.2.3 int check_mycolumns_pred (char *pred_to_set[MAX_VALUE_LEN],* int *num_columns,* int *table_num,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* int *num_pred*)

Check if the predicate has the correct matching column names.

Parameters

pred_to_set	predicate to parse
num_columns	number of columns for the table
table_num	index of the table parsing
mycolumns	names of the columns

Returns

returns true(1) if all column names match, false(-1) if it doesn't

Definition at line 514 of file server.c.

References get_param(), has_column(), MAX_VALUE_LEN, split_query_get_column(), and trim().

Referenced by parse_predicates().

4.2.2.4 int check_predicates (char *predicates[MAX_VALUE_LEN],* int *num_columns,* int *table_num,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10],* int *num_pred*)

Check if all the predicates in the string are formatted correctly.

Parameters

predicates	predicates to parse
num_columns	number of columns for the table
table_num	index of the table parsing

mycolumns	names of the columns
column_types	types of the columns
num_pred	number of predicates to parse

Returns

returns true(1) if parses correctly, false(-1) if it doesn't

Definition at line 455 of file server.c.

References get_column_type(), get_param(), MAX_COLNAME_LEN, MAX_VALUE_LEN, split_query_get_column(), split_query_get_value(), and trim().

Referenced by parse_predicates().

4.2.2.5 char* delete_command (char *key_to_delete[MAX_KEY_LEN],* int *first_empty,* int *table_num,* int *num_columns,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]*)

Delete the item in the server based on key_to_delete.

Parameters

key_to_delete	key to set
first_empty	index of the first empty spot in keys & values
table_num	index of the table parsing
num_columns	number of columns in the table
mycolumns	names of the columns

Returns

returns success string if it works (ERR_KEY_NOT_FOUND if key_to_delete DNE in keys)

Definition at line 1267 of file server.c.

References MAX_VALUE_LEN, and server_record::metadata.

Referenced by handle_command().

4.2.2.6 int get_column_index (char *column_name[MAX_COLNAME_LEN],* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* int *table_num*)

Get the index of the column based on the column name given.

Parameters

column_name	column name to find the type for
mycolumns	names of the columns
table_num	index of the table parsing

Returns

return the index of the column name given

Definition at line 906 of file server.c.

References MAX_COLUMNS_PER_TABLE.

Referenced by predicates_true().

4.2.2.7 char* get_column_type (char *column_name[MAX_COLNAME_LEN],* int *num_columns,* int *table_num,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10]*)

Get the name of the column based on name given to search for.

Parameters

column_name	column name to find the type for
num_columns	number of columns for the table
table_num	index of the table parsing
mycolumns	names of the columns
column_types	types of the columns

Returns

return string of type of the column

Definition at line 73 of file server.c.

Referenced by check_predicates().

4.2.2.8 void get_command (char *key_to_get[MAX_KEY_LEN],* char *value_to_get[MAX_VALUE_LEN],* int *first_empty,* int *table_num*)

Get the specified value based on the key_to_get.

Parameters

key_to_get	key to serach for in keys
value_to_get	value to get
first_empty	index of the first empty spot in keys & values
table_num	which table the get is being preformed on

Returns

returns the value of the search (ERR_KEY_NOT_FOUND if it DNE)

Definition at line 622 of file server.c.

References MAX_CMD_LEN, and MAX_VALUE_LEN.

Referenced by handle_command().

4.2.2.9 int handle_command (int *sock,* char *cmd[MAX_CMD_LEN],* int * *auth_var*)

Process a command from the client.

Parameters

sock	The socket connected to the client.
cmd	The command received from the client.
∗*auth_var*	variable that keeps track if client is authorized or not

Returns

Returns 0 on success, -1 otherwise.

Definition at line 1369 of file server.c.

References delete_command(), get_command(), get_param(), has_table(), key_exist(), MAX_CMD_LEN, MAX_ENC-_PASSWORD_LEN, MAX_KEY_LEN, MAX_PATH_LEN, MAX_TABLE_LEN, MAX_USERNAME_LEN, MAX_VALU-E_LEN, parse_predicates(), parse_value(), config_params::password, query_command(), sendall(), set_command(), trim(), update_command(), and config_params::username.

Referenced by main().

4.2.2.10 int has_column (char *col_to_search[MAX_COLNAME_LEN],* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABL-E][MAX_COLNAME_LEN],* int *num_columns,* int *table_num*)

Check if table exists in the server.

Parameters

col_to_search	column name to search for
mycolumns	names of all the columns of all the tables
num_columns	number of columns for the table
table_num	index of the table that is being parsed

Returns

returns column index if it exists, -1 if it doesn't

Definition at line 235 of file server.c.

Referenced by check_mycolumns_pred().

4.2.2.11 int has_table (char *table_name[MAX_TABLE_LEN]*)

Check if table exists in the server.

Parameters

table_name	table name to search for

Returns

returns table index if it exists, false(0) if it doesn't

Definition at line 210 of file server.c.

Referenced by handle_command().

4.2.2.12 int key_exist (char *key_to_search_for[MAX_KEY_LEN],* int *first_empty,* int *table_num*)

Check if key exists in the server.

Parameters

key_to_search	key to search for in server
first_empty	index of the first empty spot in keys & values
table_num	the table to search the key for

Returns

returns index of record with matching key if it exists, false(-1) if it doesn't

Definition at line 47 of file server.c.

Referenced by handle_command().

4.2.2.13 int main (int *argc,* char * *argv[]*)

Start the storage server.

This is the main entry point for the storage server. It reads the configuration file, starts listening on a port, and proccesses commands from clients.

Definition at line 1879 of file server.c.

References arguements::clientaddr_, arguements::clientaddrlen_, handle_command(), logger(), MAX_CMD_LEN, MAX_LISTENQUEUELEN, MAX_RECORDS_PER_TABLE, MAX_TABLES, read_config(), recvline(), config_params::server_host, config_params::server_port, and arguements::sock_.

4.2.2.14 int num_col_val (char *val_to_set[MAX_VALUE_LEN],* int *num_columns*)

Check if the value has the correct number of columns.

Parameters

val_to_set	value to parse
num_columns	number of columns for the table

Returns

returns true(1) if right number of columns, false(-1) if it doesn't

Definition at line 391 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by parse_value().

4.2.2.15 int num_of_predicates (char *predicates[MAX_VALUE_LEN],* int *num_columns*)

Check if the predicate string has the correct number of predicates.

Parameters

predicates	predicates to parse
num_columns	number of columns for the table

Returns

returns number of predicates if right number of predicates, false(-1) if there are too many

Definition at line 558 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by parse_predicates().

4.2.2.16 int parse_predicates (char *predicates[MAX_VALUE_LEN],* int *table_num*)

Umbrella function for all predicate parsing.

Parameters

predicates	predicates to parse
table_num	index of the table parsing
params	config parameters from config file

Returns

returns true(1) if matches all parsing, false(0) if it doesn't

Definition at line 595 of file server.c.

References check_mycolumns_pred(), check_predicates(), and num_of_predicates().

Referenced by handle_command().

4.2.2.17 int parse_value (char *val_to_set[MAX_VALUE_LEN],* int *table_num*)

Checks if the value passes all parsing.

Parameters

val_to_set	value to parse
table_num	table index for the parsing

Returns

returns true(1) if matches all parsing, false(-1) if it doesn't

Definition at line 427 of file server.c.

References check_column_types(), check_mycolumns(), and num_col_val().

Referenced by handle_command().

4.2.2.18 int predicate_true (char *predicate[MAX_VALUE_LEN],* int *table_num,* int *column_index,* int *row_index,* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10]*)

Check if the value in the row index and column index passes the predicate.

Parameters

predicate	predicate to test for true or false
table_num	index of the table parsing
column_index	index of the column to check the predicate for
row_index	index of the row to check the predicate for
column_types	types of the columns

Returns

returns true(1) if the predicate is true, false(0) if it doesn't

Definition at line 928 of file server.c.

References get_param(), MAX_VALUE_LEN, and split_query_get_value().

Referenced by predicates_true().

4.2.2.19 int predicates_true (char *predicates[MAX_VALUE_LEN],* int *num_columns,* int *table_num,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10],* int *row_index*)

Check if the value in the row index and column index passes the predicates.

Parameters

predicates	predicates to check if true or false
numcolumns	number of columns in the table
table_num	index of the table parsing
mycolumns	names of the columns
column_types	types of the columns
row_index	index of the row to check the predicate for

Returns

returns true(1) if the predicate is true, false(0) if it doesn't

Definition at line 1084 of file server.c.

References get_column_index(), get_param(), MAX_COLNAME_LEN, MAX_VALUE_LEN, predicate_true(), split_query_get_column(), and trim().

Referenced by query_command().

4.2.2.20 int query_command (int *sock,* char *predicates[MAX_VALUE_LEN],* int *first_empty,* int *table_num,* int *num_columns,* char *mycolumns[MAX_TABLES][MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN],* char *column_types[MAX_TABLES][MAX_COLUMNS_PER_TABLE][10]*)

Query the table for matching values.

Parameters

sock	The socket connected to the client.
predicates	predicates to check if true or false
first_empty	index of the first empty spot in keys & values
table_num	index of the table parsing
numcolumns	number of columns in the table
mycolumns	names of the columns
column_types	types of the columns

Returns

returns the status for the server

Definition at line 1154 of file server.c.

References MAX_CMD_LEN, MAX_RECORDS_PER_TABLE, predicates_true(), recvline(), and sendall().

Referenced by handle_command().

4.2.2.21 char* set_command (char *key_to_set[MAX_KEY_LEN],* char *value_to_set[MAX_VALUE_LEN],* int *first_empty,* int *table_num*)

Set the item using key_to_set and value_to_set.

Parameters

key_to_set	key to set
value_to_set	value to set
first_empty	index of the first empty spot in keys & values
table_num	index of the table parsing

Returns

returns success string if it works (ERR_UNKNOWN if table already at max)

Definition at line 773 of file server.c.

References MAX_RECORDS_PER_TABLE, and server_record::metadata.

Referenced by handle_command().

4.2.2.22 void split_query_get_column (char *pred[MAX_VALUE_LEN],* char *ret[MAX_VALUE_LEN]*)

Get the column name from a query predicate.

Parameters

pred	predicate to get the column out of
ret	string to return the column name too

Returns

no return value

Definition at line 103 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by check_mycolumns_pred(), check_predicates(), and predicates_true().

4.2.2.23 void split_query_get_value (char *pred[MAX_VALUE_LEN],* char *ret[MAX_VALUE_LEN]*)

Get the comparison value from a query predicate.

Parameters

pred	predicate to get the value out of
ret	string to return the column name too

Returns

no return value

Definition at line 132 of file server.c.

References get_param(), and MAX_VALUE_LEN.

Referenced by check_predicates(), and predicate_true().

4.2.2.24 char* trim (char * *str*)

Remove whitespace from beginning and end of string.

Parameters

str	string to remove leading and trailing whitespace from

Returns

trimmed version of the string

Definition at line 160 of file server.c.

Referenced by check_mycolumns_pred(), check_predicates(), handle_command(), and predicates_true().

4.2.2.25 char* update_command (char *key_to_update[MAX_KEY_LEN],* char *value_to_update[MAX_VALUE_LEN],* int *record_loc,* int *table_num,* unsigned long int *meta_data_recieved*)

Set the item using key_to_set and value_to_set.

Parameters

key_to_update	key to set
value_to_update	value to set
record_loc	index of the record to update

table_num	index of the table parsing
meta_data_recieved	meta data recieved from client

Returns

returns success string if it works (always successful, due to key_exist() check)

Definition at line 874 of file server.c.

References server_record::metadata.

Referenced by handle_command().

4.3 storage.c File Reference

This file contains the implementation of the storage server interface as specified in storage.h.

```
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include "storage.h"
#include "utils.h"
#include <stdbool.h>
```

Functions

- void * storage_connect (const char *hostname, const int port)

  Connects the client to the server.
- int storage_auth (const char *username, const char *passwd, void *conn)

  Authenticate the client via the server.
- int storage_get (const char *table, const char *key, struct storage_record *record, void *conn)

  Get a record from the server.
- int storage_set (const char *table, const char *key, struct storage_record *record, void *conn)

  Insert or delete a record from the server.
- int **storage_query** (const char *table, const char *predicates, char *keys[MAX_RECORDS_PER_TABLE], const int max_keys, void *conn)
- int storage_disconnect (void *conn)

  Connects the client to the server.

Variables

- FILE * **fclientOut**

4.3.1 Detailed Description

This file contains the implementation of the storage server interface as specified in storage.h.

Definition in file storage.c.

4.3.2 Function Documentation

4.3.2.1 int storage_auth (const char * *username*, const char * *passwd*, void * *conn*)

Authenticate the client via the server.

Authenticate the client's connection to the server.

Parameters

username	username of client
passwd	password of client
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 84 of file storage.c.

References ERR_AUTHENTICATION_FAILED, ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, generate_encrypted_password(), logger(), MAX_CMD_LEN, recvline(), and sendall().

4.3.2.2 void* storage_connect (const char * *hostname*, const int *port*)

Connects the client to the server.

Establish a connection to the server.

Parameters

hostname	hostname of the server
port	port number of the server

Definition at line 27 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_UNKNOWN, logger(), and MAX_PORT_LEN.

4.3.2.3 int storage_disconnect (void * *conn*)

Connects the client to the server.

Close the connection to the server.

Parameters

conn	represents connection to the server

Returns

0 if successful / -1 if unsuccessful

Definition at line 531 of file storage.c.

References ERR_INVALID_PARAM.

4.3.2.4 int storage_get (const char * *table,* const char * *key,* struct **storage_record** * *record,* void * *conn*)

Get a record from the server.

Retrieve the value associated with a key in a table.

Parameters

**table*	name of table being searched
**key*	key of the record being search
**record*	struct to store the value of *key
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 132 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, get_param(), logger(), MAX_CMD_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.3.2.5 int storage_set (const char * *table,* const char * *key,* struct **storage_record** * *record,* void * *conn*)

Insert or delete a record from the server.

Store a key/value pair in a table.

Parameters

**table*	name of table where record is being set
**key*	key of the record being set
**record*	struct to store the value of *key
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 245 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, ERR_TRANSACTION_ABORT, logger(), MAX_CMD_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.4 storage.h File Reference

This file defines the interface between the storage client and server.

```
#include <stdint.h>
```

Classes

- struct storage_record

  Encapsulate the value associated with a key in a table.

Macros

- #define MAX_CONFIG_LINE_LEN 1024

  Max characters in each config file line.
- #define MAX_USERNAME_LEN 64

  Max characters of server username.
- #define MAX_ENC_PASSWORD_LEN 64

  Max characters of server's encrypted password.
- #define MAX_HOST_LEN 64

  Max characters of server hostname.
- #define MAX_PORT_LEN 8

  Max characters of server port.
- #define MAX_PATH_LEN 256

  Max characters of data directory path.
- #define MAX_TABLES 100

  Max tables supported by the server.
- #define MAX_RECORDS_PER_TABLE 1000

  Max records per table.
- #define MAX_TABLE_LEN 20

  Max characters of a table name.
- #define MAX_KEY_LEN 20

  Max characters of a key name.
- #define MAX_CONNECTIONS 10

  Max simultaneous client connections.
- #define MAX_COLUMNS_PER_TABLE 10

  Max columns per table.
- #define MAX_COLNAME_LEN 20

  Max characters of a column name.
- #define MAX_STRTYPE_SIZE 40

  Max SIZE of string types.
- #define MAX_VALUE_LEN 800

  Max characters of a value.
- #define ERR_INVALID_PARAM 1

  A parameter is not valid.
- #define ERR_CONNECTION_FAIL 2

Error connecting to server.

- #define ERR_NOT_AUTHENTICATED 3

  Client not authenticated.
- #define ERR_AUTHENTICATION_FAILED 4

  Client authentication failed.
- #define ERR_TABLE_NOT_FOUND 5

  The table does not exist.
- #define ERR_KEY_NOT_FOUND 6

  The key does not exist.
- #define ERR_UNKNOWN 7

  Any other error.
- #define ERR_TRANSACTION_ABORT 8

  Transaction abort error.

Functions

- void * storage_connect (const char *hostname, const int port)

  Establish a connection to the server.
- int storage_auth (const char *username, const char *passwd, void *conn)

  Authenticate the client's connection to the server.
- int storage_get (const char *table, const char *key, struct storage_record *record, void *conn)

  Retrieve the value associated with a key in a table.
- int storage_set (const char *table, const char *key, struct storage_record *record, void *conn)

  Store a key/value pair in a table.
- int storage_query (const char *table, const char *predicates, char **keys, const int max_keys, void *conn)

  Query the table for records, and retrieve the matching keys.
- int storage_disconnect (void *conn)

  Close the connection to the server.

4.4.1 Detailed Description

This file defines the interface between the storage client and server. The functions here should be implemented in storage.c.

You should not modify this file, or else the code used to mark your implementation will break.

Definition in file storage.h.

4.4.2 Function Documentation

4.4.2.1 int storage_auth (const char * *username*, const char * *passwd*, void * *conn*)

Authenticate the client's connection to the server.

Parameters

username	Username to access the storage server.
passwd	Password in its plain text form.
conn	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to ERR_AUTHENTICATION_FAILED.

Authenticate the client's connection to the server.

Parameters

username	username of client
passwd	password of client
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 84 of file storage.c.

References ERR_AUTHENTICATION_FAILED, ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, generate_encrypted_password(), logger(), MAX_CMD_LEN, recvline(), and sendall().

4.4.2.2 void* storage_connect (const char * *hostname,* const int *port*)

Establish a connection to the server.

Parameters

hostname	The IP address or hostname of the server.
port	The TCP port of the server.

Returns

If successful, return a pointer to a data structure that represents a connection to the server. Otherwise return NULL.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, or ERR_UNKNOWN.

Establish a connection to the server.

Parameters

hostname	hostname of the server
port	port number of the server

Definition at line 27 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_UNKNOWN, logger(), and MAX_PORT_LEN.

4.4.2.3 int storage_disconnect (void * *conn*)

Close the connection to the server.

Parameters

conn	A pointer to the connection structure returned in an earlier call to storage_connect().

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, or ERR_UNKNOWN.

Close the connection to the server.

Parameters

conn	represents connection to the server

Returns

0 if successful / -1 if unsuccessful

Definition at line 531 of file storage.c.

References ERR_INVALID_PARAM.

4.4.2.4 int storage_get (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

Retrieve the value associated with a key in a table.

Parameters

table	A table in the database.
key	A key in the table.
record	A pointer to a record struture.
conn	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, ERR_TABLE_NOT_FOUND, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, or ERR_UNKNOWN.

The record with the specified key in the specified table is retrieved from the server using the specified connection. If the key is found, the record structure is populated with the details of the corresponding record. Otherwise, the record structure is not modified.

Retrieve the value associated with a key in a table.

Parameters

∗table	name of table being searched
∗key	key of the record being search
∗record	struct to store the value of ∗key
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 132 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, get_param(), logger(), MAX_CMD_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.4.2.5 int storage_query (const char * *table*, const char * *predicates*, char ** *keys*, const int *max_keys*, void * *conn*)

Query the table for records, and retrieve the matching keys.

Parameters

table	A table in the database.
predicates	A comma separated list of predicates.
keys	An array of strings where the keys whose records match the specified predicates will be copied. The array must have room for at least max_keys elements. The caller must allocate memory for this array.
max_keys	The size of the keys array.
conn	A connection to the server.

Returns

Return the number of matching keys (which may be more than max_keys) if successful, and -1 otherwise.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, ERR_TABLE_NOT_FOUND, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, or ERR_UNKNOWN.

Each predicate consists of a column name, an operator, and a value, each separated by optional whitespace. The operator may be a "=" for string types, or one of "$<$, $>$, =" for int and float types. An example of query predicates is "name = bob, mark $>$ 90".

4.4.2.6 int storage_set (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

Store a key/value pair in a table.

Parameters

table	A table in the database.
key	A key in the table.
record	A pointer to a record struture.
conn	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, ERR_TABLE_NOT_FOUND, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, or ERR_UNKNOWN.

The key and record are stored in the table of the database using the connection. If the key already exists in the table, the corresponding record is updated with the one specified here. If the key exists in the table and the record is NULL,

the key/value pair are deleted from the table.

Store a key/value pair in a table.

Parameters

∗table	name of table where record is being set
∗key	key of the record being set
∗record	struct to store the value of ∗key
conn	represents connection to the server

Returns

returns 0 if successful / -1 if unsuccessfull

Definition at line 245 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, ERR_TRANSACTION_ABORT, logger(), MAX_CMD_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.5 utils.c File Reference

This file implements various utility functions that are can be used by the storage server and client library.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include "utils.h"
#include "parser.tab.h"
```

Functions

- int **yyparse** ()
- int sendall (const int sock, const char ∗buf, const size_t len)

  Keep sending the contents of the buffer until complete.
- int recvline (const int sock, char ∗buf, const size_t buflen)

  Receive an entire line from a socket.
- int read_config (const char ∗config_file, struct config_params ∗params)

  Read and load configuration parameters.
- void logger (FILE ∗file, char ∗message, int loggingConstant)

  Generates a log message.
- char ∗ generate_encrypted_password (const char ∗passwd, const char ∗salt)

  Generates an encrypted password string using salt CRYPT_SALT.
- void get_param (char str[MAX_VALUE_LEN], char to_return[MAX_VALUE_LEN], int param_num, char ∗delims)

  Split a string and return one of the split strings.

Variables

- FILE * **yyin**
- int **error_occurred** = 0
- int **server_hostcount** = 0
- int **server_portcount** = 0
- int **usernamecount** = 0
- int **passwordcount** = 0
- int **tablecount** = 0
- int **storagepolicycount** = 0
- int **datadirectorycount** = 0
- struct config_params **paramslex**

4.5.1 Detailed Description

This file implements various utility functions that are can be used by the storage server and client library.

Definition in file utils.c.

4.5.2 Function Documentation

4.5.2.1 char* generate_encrypted_password (const char * *passwd*, const char * *salt*)

Generates an encrypted password string using salt CRYPT_SALT.

Parameters

passwd	Password before encryption.
salt	Salt used to encrypt the password. If NULL default value DEFAULT_CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 151 of file utils.c.

References DEFAULT_CRYPT_SALT.

Referenced by storage_auth().

4.5.2.2 void get_param (char *str[MAX_VALUE_LEN]*, char *to_return[MAX_VALUE_LEN]*, int *param_num*, char * *delims*)

Split a string and return one of the split strings.

Parameters

str	string to be split
to_return	place to return the specified paramter in str
param_num	which index of the split str to return
delims	string of all characters used to deliminate string

Returns

no return value

Definition at line 168 of file utils.c.

Referenced by check_column_types(), check_mycolumns(), check_mycolumns_pred(), check_predicates(), handle_command(), num_col_val(), num_of_predicates(), predicate_true(), predicates_true(), split_query_get_column(), split_query_get_value(), and storage_get().

4.5.2.3 void logger (FILE * *file,* char * *message,* int *LOGGING*)

Generates a log message.

Parameters

file	The output stream
message	Message to log.

Definition at line 131 of file utils.c.

Referenced by main(), storage_auth(), storage_connect(), storage_get(), and storage_set().

4.5.2.4 int read_config (const char * *config_file,* struct config_params * *params*)

Read and load configuration parameters.

Parameters

config_file	The name of the configuration file.
params	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Definition at line 81 of file utils.c.

References MAX_COLUMNS_PER_TABLE, config_params::password, config_params::server_host, config_params::server_port, and config_params::username.

Referenced by main().

4.5.2.5 int recvline (const int *sock,* char * *buf,* const size_t *buflen*)

Receive an entire line from a socket.

In order to avoid reading more than a line from the stream, this function only reads one byte at a time. This is very inefficient, and you are free to optimize it or implement your own function.

Definition at line 52 of file utils.c.

Referenced by main(), query_command(), storage_auth(), storage_get(), and storage_set().

4.5.2.6 int sendall (const int *sock*, const char * *buf*, const size_t *len*)

Keep sending the contents of the buffer until complete.

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the send() function.

Definition at line 32 of file utils.c.

Referenced by handle_command(), query_command(), storage_auth(), storage_get(), and storage_set().

4.6 utils.h File Reference

This file declares various utility functions that are can be used by the storage server and client library.

```
#include <stdio.h>
#include <pthread.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "storage.h"
```

Classes

- struct config_params

  A struct to store config parameters.
- struct server_record

  Encapsulate the value associated with a key in a table.
- struct arguements

  Encapsulate the value associated with a key in a table.

Macros

- #define **LOGGING_CLIENT** 1
- #define **LOGGING_SERVER** 1
- #define MAX_CMD_LEN (1024 * 8)

  The max length in bytes of a command from the client to the server.
- #define LOG(x) {printf x; fflush(stdout);}

  A macro to log some information.
- #define DBG(x) {printf x; fflush(stdout);}

  A macro to output debug information.
- #define DEFAULT_CRYPT_SALT "xx"

  Default two character salt used for password encryption.

Functions

- int sendall (const int sock, const char *buf, const size_t len)

  Keep sending the contents of the buffer until complete.
- int recvline (const int sock, char *buf, const size_t buflen)

  Receive an entire line from a socket.
- int read_config (const char *config_file, struct config_params *params)

  Read and load configuration parameters.
- void logger (FILE *file, char *message, int LOGGING)

  Generates a log message.
- char * generate_encrypted_password (const char *passwd, const char *salt)

  Generates an encrypted password string using salt CRYPT_SALT.
- void get_param (char str[MAX_VALUE_LEN], char to_return[MAX_VALUE_LEN], int param_num, char *delims)

  Split a string and return one of the split strings.

4.6.1 Detailed Description

This file declares various utility functions that are can be used by the storage server and client library.

Definition in file utils.h.

4.6.2 Macro Definition Documentation

4.6.2.1 #define DBG(*x*) {printf x; fflush(stdout);}

A macro to output debug information.

It is only enabled in debug builds.

Definition at line 47 of file utils.h.

4.6.2.2 #define LOG(*x*) {printf x; fflush(stdout);}

A macro to log some information.

Use it like this: LOG(("Hello %s", "world\n"))

Don't forget the double parentheses, or you'll get weird errors!

Definition at line 37 of file utils.h.

4.6.3 Function Documentation

4.6.3.1 char* generate_encrypted_password (const char * *passwd,* const char * *salt*)

Generates an encrypted password string using salt CRYPT_SALT.

Parameters

passwd	Password before encryption.
salt	Salt used to encrypt the password. If NULL default value DEFAULT_CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 151 of file utils.c.

References DEFAULT_CRYPT_SALT.

Referenced by storage_auth().

4.6.3.2 void get_param (char *str[MAX_VALUE_LEN],* char *to_return[MAX_VALUE_LEN],* int *param_num,* char * *delims*)

Split a string and return one of the split strings.

Parameters

str	string to be split
to_return	place to return the specified paramter in str
param_num	which index of the split str to return
delims	string of all characters used to deliminate string

Returns

no return value

Definition at line 168 of file utils.c.

Referenced by check_column_types(), check_mycolumns(), check_mycolumns_pred(), check_predicates(), handle_command(), num_col_val(), num_of_predicates(), predicate_true(), predicates_true(), split_query_get_column(), split_query_get_value(), and storage_get().

4.6.3.3 void logger (FILE * *file,* char * *message,* int *LOGGING*)

Generates a log message.

Parameters

file	The output stream
message	Message to log.

Definition at line 131 of file utils.c.

Referenced by main(), storage_auth(), storage_connect(), storage_get(), and storage_set().

4.6.3.4 int read_config (const char * *config_file,* struct config_params * *params*)

Read and load configuration parameters.

Parameters

config_file	The name of the configuration file.
params	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Definition at line 81 of file utils.c.

References MAX_COLUMNS_PER_TABLE, config_params::password, config_params::server_host, config_params::server_port, and config_params::username.

Referenced by main().

4.6.3.5 int recvline (const int *sock,* char * *buf,* const size_t *buflen*)

Receive an entire line from a socket.

Returns

Return 0 on success, -1 otherwise.

In order to avoid reading more than a line from the stream, this function only reads one byte at a time. This is very inefficient, and you are free to optimize it or implement your own function.

Definition at line 52 of file utils.c.

Referenced by main(), query_command(), storage_auth(), storage_get(), and storage_set().

4.6.3.6 int sendall (const int *sock,* const char * *buf,* const size_t *len*)

Keep sending the contents of the buffer until complete.

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the send() function.

Definition at line 32 of file utils.c.

Referenced by handle_command(), query_command(), storage_auth(), storage_get(), and storage_set().

Index