DOCS

Generated by Doxygen 1.12.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 BloomFilter Class Reference

Implements a probabilistic function with false-positive chances to check whether a key exists in a file.

```
#include <database.h>
```

Public Member Functions

- BloomFilter ()

    Constructor to initialize the class variables.
- void add (const string &key)

    Adds a key to the Bloom filter.
- bool contains (string &key)

    Checks if a key is present in the Bloom filter.
- void clear ()

    Clears the Bloom filter.

4.1.1 Detailed Description

Implements a probabilistic function with false-positive chances to check whether a key exists in a file.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BloomFilter()

```
BloomFilter::BloomFilter ()
```

Constructor to initialize the class variables.

Constructs a new Bloom filter for the database.

This constructor initializes a Bloom filter with a set number of filters.

4.1.3 Member Function Documentation

4.1.3.1 add()

```
void BloomFilter::add (
            const string & key)
```

Adds a key to the Bloom filter.

Parameters

key	The key to be added.

This function hashes the given key using multiple hash functions and sets the corresponding bits in the Bloom filter's bit array.

Parameters

key	The key to be added to the Bloom filter.

4.1.3.2 clear()

```
void BloomFilter::clear ()
```

Clears the Bloom filter.

Clears the Bloom filter's bit array.

This function resets the bit array to its initial state (empty).

4.1.3.3 contains()

```
bool BloomFilter::contains (
             string & key)
```

Checks if a key is present in the Bloom filter.

Checks whether a key is present in the Bloom filter.

Parameters

key	The key to be checked.

Returns

True if the key is present; otherwise, false.

This function checks the bits corresponding to the hash values of the given key. If any of the bits are not set, the key is not present in the Bloom filter.

Parameters

key	The key to be checked in the Bloom filter.

Returns

true If the key is possibly present in the Bloom filter, false if it is definitely not present.

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

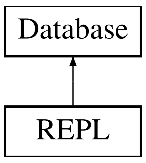
- database.h
- database.cpp

4.2 Database Class Reference

Manages data storage and retrieval, supporting operations like compaction, flushing, and concurrency control.

```
#include <database.h>
```

Inheritance diagram for Database:



Protected Member Functions

- bool Find (int tier, int index, string &key, string &value)

    Searches for a key in the database.
- bool binary_search (ifstream &In, ifstream &secondary, size_t entries, string &key, string &value)

    Performs a binary search within a file stream for a given key.
- void compact_main ()

    Main compaction process.
- void FLUSH ()

    Flushes the in-memory table to SSTable storage.
- Database ()

    Constructor for the Database class.
- ∼Database ()

    Destructor for the Database class.
- void Rename (path &oldPath, path &newPath, int tier)

    Renames files based on the nomenclature of the specified tier.
- void compact (int tier)

    Performs compaction on a specific tier.
- void write_lock (int tier)

    Locks the i-th tier for writing.
- void write_unlock (int tier)

    Unlocks the i-th tier for writing.
- void read_lock (int tier)

    Locks the i-th tier for reading.
- void read_unlock (int tier)

    Unlocks the i-th tier for reading.
- void merge_lock (int tier)

    Locks the i-th tier for merging data.
- void merge_unlock (int tier)

    Unlocks the i-th tier after merging data.
- void get_folder (int tier, path &dir)

    Retrieves the path to a specific tier directory. Creates it if it doesn't exist.
- bool initialize_folder (int tier)

    Initializes variables based on the data in a tier during database initialization.
- void initialize_memtable ()

    Initializes the in-memory table using the WAL.

- void append_to_WAL (string &key, string &value)

  Appends a key-value pair to the WAL.
- void initialize_filter (int tier, vector< BloomFilter > &filter, path &dir)

  Initializes the Bloom filters for a tier based on its data.
- void push_semaphores ()

  Pushes semaphore configurations for new tiers and threads.

Protected Attributes

- int **flushid**
- int **compactid**

  IDs for flush and compaction semaphores.
- bool **destroy**

  Indicates whether the database is being destroyed.
- vector< vector< BloomFilter > > **filters**

  Bloom filters for different tiers.
- ofstream **wal**

  Write-ahead log (WAL) file stream.
- map< string, string > **memtable**

  In-memory key-value storage.
- vector< int > **levels_main**

  Stores the number of data files in Tier_i.
- size_t **mem_size**

  Size of the in-memory table.
- vector< int > **semids**
- vector< int > **wsemids**
- vector< int > **wcount**
- vector< int > **rcount**
- vector< int > **reader**
- vector< int > **writer**
- vector< int > **mtx**

  Semaphore and thread control variables.
- bool **flushrunning**

  Flag indicating if a flush operation is in progress.
- thread **compact_main_thread**
- thread **flush_thread**

  Threads for compaction and flushing operations.

4.2.1 Detailed Description

Manages data storage and retrieval, supporting operations like compaction, flushing, and concurrency control.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Database()

```
Database::Database ()  [protected]
```

Constructor for the Database class.

Initializes the database by setting up necessary structures and background tasks.

This constructor sets up Bloom filters, initializes directories, prepares the WAL, and starts background threads for compaction and flushing. It ensures the database is ready for operation by setting the initial levels and performing any necessary compactions.

4.2.2.2 ∼Database()

```
Database::∼Database ()  [protected]
```

Destructor for the Database class.

Destructor that cleans up resources and terminates background threads.

This destructor ensures the database is properly cleaned up when it is destroyed, releasing resources such as semaphores and threads. It sets the destroy flag to stop the FLUSH and compaction threads.

4.2.3 Member Function Documentation

4.2.3.1 append_to_WAL()

```
void Database::append_to_WAL (
             string & key,
             string & val)  [protected]
```

Appends a key-value pair to the WAL.

Appends a key-value pair to the Write-Ahead Log (WAL).

Parameters

key	The key to append.
value	The value to append.

This function writes a key-value pair to the WAL, including the lengths of both the key and value, followed by the actual data.

Parameters

key	The key to be written to the WAL.
val	The value to be written to the WAL.

4.2.3.2 binary_search()

```
bool Database::binary_search (
             ifstream & In,
             ifstream & metadata,
             size_t entries,
             string & key,
             string & value)  [protected]
```

Performs a binary search within a file stream for a given key.

Performs binary search on the data file using the metadata file.

Parameters

In	The primary input file stream.
secondary	An optional secondary file stream.
entries	The number of entries in the stream.
key	The key to search for.
value	The value associated with the key, if found.

Returns

True if the key is found; otherwise, false.

This function performs a binary search over a data file using the metadata file, which contains the indexes of each string in the data file. It returns true if the key is found, false otherwise.

Parameters

In	Input file stream for the data file.
metadata	Input file stream for the metadata file.
entries	Total number of entries in the data file.
key	The key to be searched for.
value	The value associated with the key, returned if found.

Returns

true if the key was found, false otherwise.

4.2.3.3 compact()

```
void Database::compact (
             int i)  [protected]
```

Performs compaction on a specific tier.

Compacts the data in Tier_i and merges it into Tier_i+1.

Parameters

tier	The tier to compact.

This function performs the compaction of data in Tier_i, merges the files, and appends the result to Tier_i+1 as a temporary file.

Parameters

i	Index of the current tier to be compacted.

4.2.3.4 compact_main()

```
void Database::compact_main ()  [protected]
```

Main compaction process.

Main thread for calling compact operations in a loop.

This function continuously monitors the `compactid` semaphore and initiates compaction operations when required. It also ensures that the threads are properly joined before terminating the function.

4.2.3.5 Find()

```
bool Database::Find (
             int i,
             int j,
             string & key,
             string & value)  [protected]
```

Searches for a key in the database.

Does bookkeeping and calls binary_search for searching key in the file using binary search.

Parameters

tier	The tier to search in.
index	The index within the tier.
key	The key to search for.
value	The value associated with the key, if found.

Returns

True if the key is found; otherwise, false.

This function calls binary_search for looking for a specific key in the data files of a given tier and file, performs a binary search using metadata, and retrieves the corresponding value.

Parameters

i	The tier level where the data file is located.
j	The index of the specific file in the tier.
key	The key to search for.
value	The value corresponding to the key if found.

Returns

true If the key is found, false otherwise.

4.2.3.6 FLUSH()

```
void Database::FLUSH ()  [protected]
```

Flushes the in-memory table to SSTable storage.

Flushes the memtable to a new sstable on disk.

This function writes the current memtable (in-memory key-value pairs) to a new sstable on disk. It also handles writing the associated metadata and updating the Bloom filter.

4.2.3.7 get_folder()

```
void Database::get_folder (
             int i,
             path & Tier)  [protected]
```

Retrieves the path to a specific tier directory. Creates it if it doesn't exist.

Checks if the i-th tier exists, and if not, creates it and initializes necessary variables.

Parameters

tier	The tier number.
dir	The resulting directory path.

This function checks the existence of the i-th tier and initializes necessary variables if the tier does not exist.

Parameters

i	Index of the tier.
Tier	Reference to the tier path.

4.2.3.8 initialize_filter()

```
void Database::initialize_filter (
             int i,
             vector< BloomFilter > & filter,
             path & Tier)  [protected]
```

Initializes the Bloom filters for a tier based on its data.

Initializes the BloomFilter for the i-th tier.

Parameters

tier	The tier number.
filter	The Bloom filter vector to initialize.
dir	The directory path of the tier.

This function initializes the BloomFilter for the i-th tier by loading data from disk. It reads both data files and their corresponding metadata.

Parameters

i	Index of the tier.
filter	Reference to the BloomFilter vector for the tier.
Tier	Reference to the path of the tier.

4.2.3.9 initialize_folder()

```
bool Database::initialize_folder (
             int i)  [protected]
```

Initializes variables based on the data in a tier during database initialization.

Initializes the folder structure for the i-th tier based on existing data files.

Parameters

tier	The tier number.

Returns

True if initialization succeeds; otherwise, false.

This function checks if the folder for the i-th tier exists and initializes it with the necessary data files and BloomFilters.

Parameters

i	Index of the tier.

Returns

true if the tier was successfully initialized, false otherwise.

4.2.3.10 initialize_memtable()

```
void Database::initialize_memtable ()  [protected]
```

Initializes the in-memory table using the WAL.

Initializes the memtable by loading data from the Write-Ahead Log (WAL).

This function reloads the memtable from the WAL file during database initialization.

4.2.3.11 merge_lock()

```
void Database::merge_lock (
             int i)  [protected]
```

Locks the i-th tier for merging data.

This function locks the semaphore for merging data in the i-th tier.

Parameters

i	Index of the tier.

4.2.3.12 merge_unlock()

```
void Database::merge_unlock (
             int i)  [protected]
```

Unlocks the i-th tier after merging data.

This function releases the lock for merging data in the i-th tier.

Parameters

i	Index of the tier.

4.2.3.13 push_semaphores()

```
void Database::push_semaphores ()  [protected]
```

Pushes semaphore configurations for new tiers and threads.

Pushes new semaphores for managing writers, readers, and mergers.

This function initializes and pushes new semaphores for managing operations in the database. These semaphores are used for mutual exclusion and synchronization of writing, reading, and merging. < Pushes semaphores for managing mergers, writers, readers

< Stores the number of writers and readers respectively

< Check if the obtained semaphores are valid

4.2.3.14 read_lock()

```
void Database::read_lock (
             int i)  [protected]
```

Locks the i-th tier for reading.

This function provides mutual exclusion between readers and ensures that if there are readers, writers will be blocked.

Parameters

i	Index of the tier.

< If there is at least one reader, wait for writers to complete

4.2.3.15 read_unlock()

```
void Database::read_unlock (
             int i)  [protected]
```

Unlocks the i-th tier for reading.

This function releases the read lock on the i-th tier and signals writers if there are no readers left.

Parameters

i	Index of the tier.

< If there are no readers, signal to writers

4.2.3.16 Rename()

```
void Database::Rename (
            path & folder,
            path & folder1,
            int i)  [protected]
```

Renames files based on the nomenclature of the specified tier.

Renames the temporary files to actual files and increments the `levels_main`.

Parameters

oldPath	The current file path.
newPath	The new file path.
tier	The tier number.

This function renames the temporary data and metadata files to their actual names and increments the `levels↩ _main` for the specified tier.

Parameters

folder	Path to the source folder containing temporary files.
folder1	Path to the destination folder.
i	Index of the tier.

4.2.3.17 write_lock()

```
void Database::write_lock (
            int i)  [protected]
```

Locks the i-th tier for writing.

This function provides mutual exclusion between multiple writers to the i-th tier. It waits for any ongoing reader operations to complete if this is the first writer.

Parameters

i	Index of the tier.

< Mutual exclusion between writers

< If there is at least one writer, wait for readers to complete

< Lock for writers

4.2.3.18 write_unlock()

```
void Database::write_unlock (
              int i ) [protected]
```

Unlocks the i-th tier for writing.

This function releases the write lock on the i-th tier. It updates the number of writers and signals the readers if there are no writers left.

Parameters

i	Index of the tier.

< If there are no writers, signal to readers

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

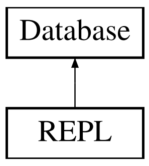
- database.h
- database.cpp

4.3 REPL Class Reference

A class derived from Database that implements basic REPL operations (GET, SET, DELETE) for the key-value store.

```
#include <REPL.h>
```

Inheritance diagram for REPL:



Public Member Functions

- bool GET (string &key, string &value)

    Retrieves the value associated with a given key.
- bool SET (string &key, string &value)

    Sets the value for a given key.
- bool DELETE (string &key)

    Marks a key as deleted by setting its value to TOMBSTONE.

Additional Inherited Members

Protected Member Functions inherited from Database

- bool Find (int tier, int index, string &key, string &value)

  Searches for a key in the database.
- bool binary_search (ifstream &In, ifstream &secondary, size_t entries, string &key, string &value)

  Performs a binary search within a file stream for a given key.
- void compact_main ()

  Main compaction process.
- void FLUSH ()

  Flushes the in-memory table to SSTable storage.
- Database ()

  Constructor for the Database class.
- ∼Database ()

  Destructor for the Database class.
- void Rename (path &oldPath, path &newPath, int tier)

  Renames files based on the nomenclature of the specified tier.
- void compact (int tier)

  Performs compaction on a specific tier.
- void write_lock (int tier)

  Locks the i-th tier for writing.
- void write_unlock (int tier)

  Unlocks the i-th tier for writing.
- void read_lock (int tier)

  Locks the i-th tier for reading.
- void read_unlock (int tier)

  Unlocks the i-th tier for reading.
- void merge_lock (int tier)

  Locks the i-th tier for merging data.
- void merge_unlock (int tier)

  Unlocks the i-th tier after merging data.
- void get_folder (int tier, path &dir)

  Retrieves the path to a specific tier directory. Creates it if it doesn't exist.
- bool initialize_folder (int tier)

  Initializes variables based on the data in a tier during database initialization.
- void initialize_memtable ()

  Initializes the in-memory table using the WAL.
- void append_to_WAL (string &key, string &value)

  Appends a key-value pair to the WAL.
- void initialize_filter (int tier, vector< BloomFilter > &filter, path &dir)

  Initializes the Bloom filters for a tier based on its data.
- void push_semaphores ()

  Pushes semaphore configurations for new tiers and threads.

Protected Attributes inherited from Database

- int **flushid**
- int **compactid**

    IDs for flush and compaction semaphores.
- bool **destroy**

    Indicates whether the database is being destroyed.
- vector< vector< BloomFilter > > **filters**

    Bloom filters for different tiers.
- ofstream **wal**

    Write-ahead log (WAL) file stream.
- map< string, string > **memtable**

    In-memory key-value storage.
- vector< int > **levels_main**

    Stores the number of data files in Tier_i.
- size_t **mem_size**

    Size of the in-memory table.
- vector< int > **semids**
- vector< int > **wsemids**
- vector< int > **wcount**
- vector< int > **rcount**
- vector< int > **reader**
- vector< int > **writer**
- vector< int > **mtx**

    Semaphore and thread control variables.
- bool **flushrunning**

    Flag indicating if a flush operation is in progress.
- thread **compact_main_thread**
- thread **flush_thread**

    Threads for compaction and flushing operations.

4.3.1 Detailed Description

A class derived from Database that implements basic REPL operations (GET, SET, DELETE) for the key-value store.

This class extends the Database class and provides methods to interact with the database through a REPL (Read-Eval-Print Loop). The methods allow retrieving, setting, and deleting key-value pairs from the database.

4.3.2 Member Function Documentation

4.3.2.1 DELETE()

```
bool REPL::DELETE (
             string & key)
```

Marks a key as deleted by setting its value to TOMBSTONE.

Marks a key as deleted by setting it to TOMBSTONE.

This function is used to delete a key from the database by setting the associated value to a special "TOMBSTONE" value. This does not immediately remove the key from the database but marks it for deletion.

Parameters

key	The key to be marked as deleted.

Returns

True if the operation was successful, false otherwise.

This function is a shortcut for calling `SET` with the TOMBSTONE value, marking the given key as deleted in the database. The value associated with the key will be set to TOMBSTONE, indicating that the key has been logically deleted.

Parameters

key	The key to be deleted.

Returns

true if the delete operation is successful, false otherwise.

4.3.2.2 GET()

```
bool REPL::GET (
            string & key,
            string & value)
```

Retrieves the value associated with a given key.

Retrieves the value for a given key from the database.

This function looks for the key in the memtable's BloomFilter and then searches for it in the memtable or disk if necessary. If the key is found and not marked as deleted (TOMBSTONE), the corresponding value is returned.

Parameters

key	The key for which the value needs to be retrieved.
value	The value corresponding to the given key (output parameter).

Returns

True if the key was found and the value is valid, false otherwise.

This function checks if the given key exists in the Bloom filter and searches for the key in the memtable. If not found in the memtable, it searches through the levels of the database (using Bloom filters and the `Find` function). If the key is found and is not marked as a TOMBSTONE, the corresponding value is returned.

Parameters

key	The key to search for.
value	The value associated with the key, to be returned if found.

Returns

true if the key is found and is not marked as a TOMBSTONE, false otherwise.

4.3.2.3 SET()

```
bool REPL::SET (
            string & key,
            string & value)
```

Sets the value for a given key.

Sets the value for a given key in the memtable and appends it to the WAL.

This function stores the key-value pair in the memtable and appends it to the Write-Ahead Log (WAL). If the memtable's size exceeds the threshold, a flush operation is triggered to persist the data on disk.

Parameters

key	The key to be inserted into the database.
value	The value associated with the key.

Returns

True if the key-value pair was successfully inserted, false otherwise (e.g., if the total size is too large).

This function adds a key-value pair to the memtable and appends the same to the Write-Ahead Log (WAL). If the memtable exceeds a certain size, it triggers a flush operation. Additionally, it ensures that the key-value pair does not exceed the maximum allowed size.

Parameters

key	The key to be set.
value	The value to be associated with the key.

Returns

true if the operation is successful, false if the key-value pair exceeds the size limit.

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

- REPL.h
- REPL.cpp

Chapter 5

File Documentation

5.1 database.h

```
#include<bits/stdc++.h>
#include<sys/sem.h>
#include<filesystem>
#include<bitset>
#define MAX 4000000
#define MIN_TH 4
#define MAX_TH 12
#define NOFILTERS 3
using namespace std;
using namespace filesystem;
using std::atomic;
#ifndef HI
#define HI
extern struct sembuf vop, pop, top;
#define V(semid) semop(semid, &vop, 1)
#define P(semid) semop(semid, &pop, 1)
#define T(semid) semop(semid, &top, 1)
extern string empty_string, TOMBSTONE;

class BloomFilter {
private:
    vector<hash<string>> filter;
    bitset<10000> bitArray;

public:
    BloomFilter();

    void add(const string& key);

    bool contains(string& key);

    void clear();
};

class Database {
protected:
    int flushid, compactid;
    bool destroy;
    vector<vector<BloomFilter>> filters;
    ofstream wal;
    map<string, string> memtable;
    vector<int> levels_main;
    size_t mem_size;
    vector<int> semids, wsemids, wcount, rcount, reader, writer, mtx;
    bool flushrunning;
    thread compact_main_thread, flush_thread;

    bool Find(int tier, int index, string& key, string& value);

    bool binary_search(ifstream& In, ifstream& secondary, size_t entries, string& key, string& value);

    void compact_main();

    void FLUSH();

    Database();

    ~Database();

    void Rename(path& oldPath, path& newPath, int tier);

    void compact(int tier);

    void write_lock(int tier);
    void write_unlock(int tier);
    void read_lock(int tier);
    void read_unlock(int tier);
    void merge_lock(int tier);
    void merge_unlock(int tier);

    void get_folder(int tier, path& dir);

    bool initialize_folder(int tier);

    void initialize_memtable();

    void append_to_WAL(string& key, string& value);

    void initialize_filter(int tier, vector<BloomFilter>& filter, path& dir);

    void push_semaphores();
};

#endif
```

5.2 REPL.h

```
#include<bits/stdc++.h>
#include"database.h"
#ifndef yu
#define yu
class REPL : public Database {
public:
    bool GET(string &key, string &value);

    bool SET(string &key, string &value);

    bool DELETE(string &key);
};
#endif
```

Index