dbHandler for sqlite3

Generated by Doxygen 1.9.0

1 README 1

1.0.1 OSS Buils Status :computer: 1

1.1 Welcome to the Sqlite3 Handler Repository! :notebook: 1

1.1.1 Announcements :clipboard: 1

1.1.2 Description :scroll: 1

1.1.3 Installation :floppy_disk: 2

1.1.4 :checkered_flag: Final Notes :checkered_flag: 2

2 Namespace Index 3

2.1 Namespace List 3

3 Class Index 5

3.1 Class List 5

4 Namespace Documentation 7

4.1 handler Namespace Reference 7

4.1.1 Detailed Description 7

4.1.2 Typedef Documentation 7

4.1.2.1 DbTables 7

4.1.2.2 FieldDescription 8

4.2 query Namespace Reference 8

4.2.1 Detailed Description 8

4.3 query::affinity Namespace Reference 8

4.3.1 Detailed Description 9

4.4 query::cl Namespace Reference 9

4.4.1 Detailed Description 9

4.4.2 Function Documentation 10

4.4.2.1 glob() 10

4.4.2.2 like() 10

4.4.2.3 limit() 10

4.4.2.4 offset() 11

4.4.2.5 table_info() 11

4.4.2.6 type() 11

4.5 query::cmd Namespace Reference 13

4.5.1 Detailed Description 14

4.6 query::data Namespace Reference 14

4.6.1 Detailed Description 14

4.6.2 Function Documentation 14

4.6.2.1 len() 14

4.6.2.2 trigger() 15

4.6.2.3 view() 15

5 Class Documentation 17

5.1 handler::select_query_param Struct Reference 17

5.1.1 Detailed Description . . . 17
5.1.2 Member Data Documentation . . . 17
5.1.2.1 fields . . . 17
5.1.2.2 group_by . . . 18
5.1.2.3 having_cond . . . 18
5.1.2.4 limit . . . 18
5.1.2.5 offset . . . 18
5.1.2.6 order_by . . . 18
5.1.2.7 order_type . . . 18
5.1.2.8 select_distinct . . . 18
5.1.2.9 table_name . . . 18
5.1.2.10 where_cond . . . 19
5.2 handler::Sqlite3Db Class Reference . . . 19
5.2.1 Detailed Description . . . 20
5.2.2 Constructor & Destructor Documentation . . . 20
5.2.2.1 Sqlite3Db() [1/2] . . . 20
5.2.2.2 Sqlite3Db() [2/2] . . . 21
5.2.2.3 ~Sqlite3Db() . . . 21
5.2.3 Member Function Documentation . . . 21
5.2.3.1 callback() . . . 21
5.2.3.2 closeConnection() . . . 22
5.2.3.3 connectDb() . . . 22
5.2.3.4 createTable() . . . 22
5.2.3.5 deleteRecords() . . . 23
5.2.3.6 dropTable() . . . 24
5.2.3.7 executeQuery() . . . 24
5.2.3.8 getAffinity() . . . 25
5.2.3.9 getDbPath() . . . 25
5.2.3.10 getFields() . . . 26
5.2.3.11 getNumTables() . . . 26
5.2.3.12 getTables() . . . 26
5.2.3.13 getTablesNames() . . . 27
5.2.3.14 insertRecord() . . . 27
5.2.3.15 isAffined() . . . 28
5.2.3.16 isInt() . . . 28
5.2.3.17 isReal() . . . 28
5.2.3.18 isValidInt() . . . 29
5.2.3.19 isValidReal() . . . 29
5.2.3.20 selectRecords() [1/2] . . . 29
5.2.3.21 selectRecords() [2/2] . . . 31
5.2.3.22 updateHandler() . . . 32

Chapter 1

README

1.0.1 OSS Buils Status :computer:

:heavy_check_mark:

1.1 Welcome to the Sqlite3 Handler Repository! :notebook:

1.1.1 Announcements :clipboard:

1.1.1.0.1 :pushpin: Actual release version This is the first release of this project (1.0.X)

1.1.1.0.2 :pushpin: Future Additions :construction: As only the so called "basic functionality" of sqlite3 is implemented at the moment, all the advanced methods will be included in the future.

The terms "basic" and "advanced" have been extracted from the `Tutorials Point page` referenced to the C++ interface of sqlite3.

Also, the inclusion of `sqlite3_bind()` for internal parameters, and `sqlite3_reset()` for the usage of commands without needing to prepare them again, is planned.

1.1.2 Description :scroll:

This project is based, and depends on, the sqlite3 official libraries. Check the **`SQLite3 website`** for more information about them, and how they can be installed on your system.

The objective of this project is to provide an straight-forward, parameterized approach on sqlite3 database management, using the **C++** interface provided from sqlite3. The idea is to apply the main functionality of the database management without the need of writing a query in plain text.

With this project you receive a set of two .hpp libraries, consisting of:

1. *query.hpp*: An aggregate of sqlite3 query commands and clauses, for you to use on your queries definition. It uses different name spaces to differentiate between command, clauses and conditions.
2. *handler.hpp*: Contains a handler class which can execute it's different methods on the database selected.

The doxygen generated documentation of the software is included in the `docs/refman.pdf`, containing the list of variables available in query.hpp, and the description of all the methods in handler.hpp. An html version of the documentation is also available in the `docs` folder.

Each of the methods available at the moment have a pseudo code example that can be seen in the `examples folder`.

1.1.2.0.1 Custom method executeQuery() :information_source:

In case the use of any query not currently supported by the methods implemented, executeQuery() can operate with any (correctly built) sqlite3 query. Check the docs for more information about how to use this method, the arguments it can take, and the information it can provide.

1.1.3 Installation :floppy_disk:

IMPORTANT NOTE :memo: This process has not been completely tested on windows yet.

The installation is managed with cmake, which you should already have installed on your system. The minimum supported version is 3.11, but some functionalities need at least the 3.18 one. If you try to execute the installer without having the proper version the process will abort, so nothing will break on your machine.

For installation it is recommended for you to create a _"build/"_ directory inside of the project source folder, then from there run "_cmake <options> .._". After that is done, execute "_cmake --build ._", and finally "_cmake --install ._", where you may want to add an installation prefix. Please take a look at this tutorial for more information about the native options you can add to each instruction.

Now lets take a look at what does the installer do in the default mode and the different options that can be added to apply more changes.

1.1.3.0.1 Generation flags :wrench:

When calling cmake .. for generating the build files, there are some option flags that you can set for customizing your installation.

- **-DINSTALL_DEPENDENCIES=ON** - This will ask the installer to install sqlite3 in your system if you do not have it already. If sqlite3 is found this option will not change the generating process. **NOTE: this process requires some cmake functionalities only available from the 3.18 version. If you do not have this or a later version, the generation process will fail. On windows this will also require nmake.**
- **-DUNIT_TESTS=ON** - This option downloads the google test framework locally, and uses it's source code to build a test suite, that could be executed using the ctest executable generated inside of the build directory. This tests ensure that the methods included in the handler library work as expected, but they are not necessary for you to build.If you do so, the tests.cpp file will also be installed on your system, so it may be interesting if you would like to develop this project further on.
- **-DINSTALL_EXAMPLES=ON** - If this is set to ON, the examples folder will be installed to the installation prefix. This examples are used as documentation, as well as a simple introduction to the methods available inside of the handler library.
- **-DINSTALL_DOCS=ON** - The documentation is generated both in pdf and in html format. Both of them will be installed if this option is set to ON. This README will also be installed with them.

1.1.3.0.2 Installation Options :cd:

- **System installation**: This installs the libraries to the system, using the location in which all C++ includes are stored. This way you will be able to use this libraries as #include <library_name.hpp> instead of using the relative or absolute path of it using #include "<path to library>/library_name.hpp".
- **Prefix Installation**: The libraries and their source files will be installed at the given path, at an **include** and **lib** folders respectively. This installation could be interesting if you would like to use them as a part of other project without having them on your system.

For both of them, all the other options you may have activated will be installed at the provided path.

1.1.4 :checkered_flag: Final Notes :checkered_flag:

The installation folder is include/sqliteutils-<version_number>/library files. So your includes should be #include <sqliteutils-x.x.x/library_name>

The provided libraries can operate in parallel or separately from the sqlite3 C++ official interface without any issue.

Chapter 2

Namespace Index

2.1 Namespace List

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

handler
Contains the Sqlite3Db class and it's types 7
query
Contains the macros for queries definitons 8
query::affinity
The data affinity definitions are stored in this namespace 8
query::cl
List including all the clauses available in the sqlite3 syntax 9
query::cmd
List of the commands in the sqlite3 syntax 13
query::data
Includes some of the different datatypes and database selectable parts 14

Chapter 3

Class Index

3.1 Class List

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

handler::select_query_param 17
handler::Sqlite3Db
Class for handling connection and operations in a sqlite3 database 19

Chapter 4

Namespace Documentation

4.1 handler Namespace Reference

Contains the Sqlite3Db class and it's types.

Classes

- struct select_query_param
- class Sqlite3Db

  Class for handling connection and operations in a sqlite3 database.

Typedefs

- typedef std::map< const std::string, std::vector< std::string > > DbTables
- typedef std::pair< std::string, std::string > FieldDescription

4.1.1 Detailed Description

Contains the Sqlite3Db class and it's types.

Namespace used to contain the Sqlite3Db class and all of it's members. It also uses some type definitions to make the software cleaner.

4.1.2 Typedef Documentation

4.1.2.1 DbTables

```
typedef std::map<const std::string, std::vector<std::string> > handler::DbTables
```

Type that stores the information of the tables inside of the db.

4.1.2.2 FieldDescription

```
typedef std::pair<std::string, std::string> handler::FieldDescription
```

For use when defining a field in the create table statement

4.2 query Namespace Reference

Contains the macros for queries definitons.

Namespaces

- affinity

  The data affinity definitions are stored in this namespace.
- cl

  List including all the clauses available in the sqlite3 syntax.
- cmd

  List of the commands in the sqlite3 syntax.
- data

  Includes some of the different datatypes and database selectable parts.

Variables

- const std::string **end_query** =";"

4.2.1 Detailed Description

Contains the macros for queries definitons.

4.3 query::affinity Namespace Reference

The data affinity definitions are stored in this namespace.

Variables

- const std::string **integer** = "INTEGER"
- const std::string **int_affinity** = "INT"
- const std::string **text** = "TEXT"
- const std::vector< std::string > **text_affinity** = {"CHAR", "CLOB", "TEXT"}
- const std::string **blob** = "BLOB"
- const std::string **blob_affinity** = "BLOB"
- const std::string **real** = "REAL"
- const std::vector< std::string > **real_affinity** = {"REAL", "FLOA", "DOUB"}
- const std::string **numeric** = "NUMERIC"

4.3.1 Detailed Description

The data affinity definitions are stored in this namespace.

4.4 query::cl Namespace Reference

List including all the clauses available in the sqlite3 syntax.

Functions

- const std::string glob (const std::string pattern)
  generates a GLOB sqlite3 clause with the pattern given.
- const std::string like (const std::string pattern)
  generates a LIKE sqlite3 clause with the pattern given.
- const std::string limit (int limit_value)
  generates a LIMIT sqlite3 clause with the value given.
- const std::string offset (int offset_value)
  generates an OFFSET sqlite3 clause with the value given.
- const std::string table_info (const std::string table_name)
  Generates a table_info clause used in PRAGMA statements.
- const std::string type (const std::string type)
  Generates a type=" clause for WHERE clause.

Variables

- const std::string **and_** = " AND "
- const std::string **as** = " AS "
- const std::string **before** = " BEFORE "
- const std::string **between** = " BETWEEN "
- const std::string **count** = " COUNT "
- const std::string **distinct** = " DISTINCT "
- const std::string **exists** = " EXISTS "
- const std::string **for_** = " FOR "
- const std::string **for_each** = " FOR EACH "
- const std::string **from** = " FROM "
- const std::string **group_by** = " GROUP BY "
- const std::string **having** = " HAVING "
- const std::string **in** = " IN "
- const std::string **not_** = " NOT "
- const std::string **on** = " ON "
- const std::string **or_** = " OR "
- const std::string **order_by** = " ORDER BY "
- const std::string **sum** = " SUM "
- const std::string **values** = " VALUES "
- const std::string **where** = " WHERE "

4.4.1 Detailed Description

List including all the clauses available in the sqlite3 syntax.

4.4.2 Function Documentation

4.4.2.1 glob()

```
const std::string query::cl::glob (
            const std::string pattern )
```

generates a GLOB sqlite3 clause with the pattern given.

Parameters

pattern	The pattern to be included. GLOB patterns use UNIX type wildcards (∗ and ?). Some examples of patterns could be 'XXXX∗', '*XXXX*', '?XXXX?', '????'...

Returns

The composed GLOB clause. "GLOB '[pattern]'"

4.4.2.2 like()

```
const std::string query::cl::like (
            const std::string pattern )
```

generates a LIKE sqlite3 clause with the pattern given.

Parameters

pattern	The pattern to be included. It's wildcards consist of '' and '_'. Some examples of patterns could be 'XXXX', 'XXXX', '*XXXX*', '____'...

Returns

The composed LIKE clause. "LIKE '[pattern]'"

4.4.2.3 limit()

```
const std::string query::cl::limit (
            int limit_value )
```

generates a LIMIT sqlite3 clause with the value given.

Parameters

limit_value	Integer number defining the limit.

Returns

The composed LIMIT clause. "LIMIT [limit_value]"

4.4.2.4 offset()

```
const std::string query::cl::offset (
            int offset_value )
```

generates an OFFSET sqlite3 clause with the value given.

Parameters

offset_value	Integer number defining the offset from where a limit clause could be applied.

Returns

The composed OFFSET clause. "OFFSET [offset_value]"

4.4.2.5 table_info()

```
const std::string query::cl::table_info (
            const std::string table_name )
```

Generates a table_info clause used in PRAGMA statements.

Parameters

table_name	The name of the table that the information is needed from.

Returns

The composed table_info clause. "table_info([table_name])"

4.4.2.6 type()

```
const std::string query::cl::type (
            const std::string type )
```

Generates a type=" clause for WHERE clause.

Parameters

type	The type that will be added to the clause.

Returns

The composed type clause. "type='[type]'"

4.5 query::cmd Namespace Reference

List of the commands in the sqlite3 syntax.

Variables

- const std::string **add_column** = " ADD COLUMN "
- const std::string **alter_table** = " ALTER TABLE "
- const std::string **analyze** = " ANALYZE "
- const std::string **attach_db** = " ATTACH DATABASE "
- const std::string **begin** = " BEGIN "
- const std::string **begin_txn** = " BEGIN EXCLUSIVE TRANSACTION "
- const std::string **commit** = " COMMIT "
- const std::string **create** = " CREATE "
- const std::string **create_uniq_indx** = " CREATE UNIQUE INDEX"
- const std::string **create_table** = " CREATE TABLE "
- const std::string **create_trigger** = " CREATE TRIGGER "
- const std::string **create_view** = " CREATE VIEW "
- const std::string **create_virtual_tbl** = " CREATE VIRTUAL TABLE "
- const std::string **delete_** = " DELETE "
- const std::string **detach_db** = " DETACH DATABASE "
- const std::string **drop_indx** = " DROP INDEX "
- const std::string **drop_table** = " DROP TABLE "
- const std::string **drop_trigger** = " DROP TRIGGER "
- const std::string **drop_view** = " DROP VIEW "
- const std::string **explain** = " EXPLAIN "
- const std::string **insert** = " INSERT "
- const std::string **insert_into** = " INSERT INTO "
- const std::string **insert_on** = " INSERT ON "
- const std::string **pragma** = " PRAGMA "
- const std::string **reindex** = " REINDEX "
- const std::string **release_savepoint** = " RELEASE "
- const std::string **rename_to** = " RENAME TO "
- const std::string **rollback** = " ROLLBACK"
- const std::string **rollback_savepoint** = " ROLLBACK TO SAVEPOINT "
- const std::string **savepoint** = " SAVEPOINT "
- const std::string **select** = " SELECT "
- const std::string **update** = " UPDATE "
- const std::string **using_** = " USING "
- const std::string **vacuum** = " VACUUM"

4.5.1 Detailed Description

List of the commands in the sqlite3 syntax.

4.6 query::data Namespace Reference

Includes some of the different datatypes and database selectable parts.

Functions

- const std::string len (int length)

  Generates sqlite3 definition of length for a datatype.
- const std::string trigger (const std::string db_name, const std::string trigger_name)

  Generates a trigger definition with the database and trigger name given.
- const std::string view (const std::string db_name, const std::string view_name)

  Generates a view definition with the database and view name given.

Variables

- const std::string **char_** = " CHAR "
- const std::string **int_** = " INT "
- const std::string **not_null** = " NOT NULL "
- const std::string **null** = " NULL "
- const std::string **primary_key** = " PRIMARY KEY "
- const std::string **row** = " ROW "
- const std::string **table** = " TABLE "

4.6.1 Detailed Description

Includes some of the different datatypes and database selectable parts.

4.6.2 Function Documentation

4.6.2.1 len()

```
const std::string query::data::len (
            int length )
```

Generates sqlite3 definition of length for a datatype.

Parameters

length	Integer number to be defined as length limit.

Returns

The composed length for a datatype. "([length])"

4.6.2.2 trigger()

```
const std::string query::data::trigger (
            const std::string db_name,
            const std::string trigger_name )
```

Generates a trigger definition with the database and trigger name given.

Parameters

db_name	Name of the database where the trigger is located.
trigger_name	Name of the trigger.

Returns

The trigger declaration. "db_name.trigger_name"

4.6.2.3 view()

```
const std::string query::data::view (
            const std::string db_name,
            const std::string view_name )
```

Generates a view definition with the database and view name given.

Parameters

db_name	Name of the database where the view is located.
view_name	Name of the view.

Returns

The view declaration. "db_name.view_name"

Chapter 5

Class Documentation

5.1 handler::select_query_param Struct Reference

`#include <handler.hpp>`

Public Attributes

- std::string table_name
- std::vector< std::string > fields = {"*"}
- bool select_distinct = false
- std::string where_cond = ""
- std::vector< std::string > group_by = {}
- std::string having_cond = ""
- std::vector< std::string > order_by = {}
- std::string order_type = "ASC"
- int limit = 0
- int offset = 0

5.1.1 Detailed Description

Structure used for storing all options that may be used during a select query.

5.1.2 Member Data Documentation

5.1.2.1 fields

`std::vector<std::string> handler::select_query_param::fields = {"*"}`

Fields to be shown in the result

5.1.2.2 group_by

```
std::vector<std::string> handler::select_query_param::group_by = {}
```

Fields grouped in the result

5.1.2.3 having_cond

```
std::string handler::select_query_param::having_cond = ""
```

Having filter condition to be applied

5.1.2.4 limit

```
int handler::select_query_param::limit = 0
```

Maximum number of results to be processed

5.1.2.5 offset

```
int handler::select_query_param::offset = 0
```

Starting point in the results to apply the limit quantity

5.1.2.6 order_by

```
std::vector<std::string> handler::select_query_param::order_by = {}
```

Fields or conditions to order the results

5.1.2.7 order_type

```
std::string handler::select_query_param::order_type = "ASC"
```

Type of ordering of the results "ASC" or "DESC"

5.1.2.8 select_distinct

```
bool handler::select_query_param::select_distinct = false
```

Flag for showing only distinct results

5.1.2.9 table_name

```
std::string handler::select_query_param::table_name
```

Name of the table in which the select query will be applied

5.1.2.10 where_cond

```
std::string handler::select_query_param::where_cond = ""
```

Condition to be applied with a WHERE clause

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

- include/handler.hpp

5.2 handler::Sqlite3Db Class Reference

Class for handling connection and operations in a sqlite3 database.

```
#include <handler.hpp>
```

Public Member Functions

- Sqlite3Db ()
  Constructor for handler using test.db.
- Sqlite3Db (std::string db_path)
  Constructor for user defined database name.
- ~Sqlite3Db ()
  Destructor of the class Sqlite3Db.
- void closeConnection ()
  Close connection to the current database.
- bool createTable (std::string table_name, std::vector< FieldDescription > fields)
  Create a table in the database with the specified parameters.
- bool deleteRecords (std::string table_name, std::string condition)
  Delete the records from a table that meet the provided condition/s.
- bool dropTable (std::string table_name)
  Drop the table specified.
- bool executeQuery (const char *sql_query, std::vector< std::string > &data=empty_vec, std::vector< int > indexes_stmt={}, bool verbose=false)
  Execute an SQLite query and receive the output selected.
- bool insertRecord (std::string table_name, std::vector< std::string > values)
  Insert record data inside of a table.
- bool connectDb ()
  Reconnects the handler to it's linked database.
- std::vector< std::string > selectRecords (std::string table_name, std::vector< std::string > fields={"*"}, bool select_distinct=false, std::string where_cond="", std::vector< std::string > group_by={}, std::string having_cond="", std::vector< std::string > order_by={}, std::string order_type="ASC", int limit=0, int offset=0)
  Selects and extracts the records that meet certain conditions.
- std::vector< std::string > selectRecords (select_query_param select_options)
  Selects and extracts the records that meet certain conditions.
- bool updateHandler ()
  Updates the information contained in the handler.
- const std::string getAffinity (const std::string field_datatype)
  Calculate the affinity token corresponding to a datatype given.

- std::vector< std::string > getFields (std::string table_name)

  Get field's names from a table in the database.
- int getNumTables ()

  Get number of tables in the database.
- DbTables getTables ()

  Get tables information map stored in the handler.
- std::vector< std::string > getTablesNames ()

  Get table's names from the database.
- std::string getDbPath ()

  Gets db relative path from the database.
- bool isAffined (const std::string affinity, const std::string value_to_check)

  Compares the value of some field to it's corresponding affinity.
- bool isValidInt (const std::string &str)

  Checks if a string is a valid integer number.
- bool isValidReal (const std::string &str)

  Checks if a string is a valid real number.

Static Public Member Functions

- static int callback (void *NotUsed, int argc, char **argv, char **azColName)

  Callback to show the output of sqlite3_exec()
- static bool isReal (char c)

  Check if a character is valid for being inside of a real number.
- static bool isInt (char c)

  Check if a character is valid for being inside of an integer number.

Friends

- std::ostream & **operator**<< (std::ostream &output, const Sqlite3Db &sqlite3Db)

5.2.1 Detailed Description

Class for handling connection and operations in a sqlite3 database.

This class contains all of the basic operations available in the sqlite3 syntax, including a custom executeQuery(), so that the user can pass their own querys and obtain the data from them, if preferred over the provided functionality.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Sqlite3Db() [1/2]

```
handler::Sqlite3Db::Sqlite3Db ( )
```

Constructor for handler using test.db.

When creating a new object, the connection to the database is attempted. If no database exists with the name provided, it is created.

Assuming the database already exists and contains some information, this constructor will load all the names of the tables present in the database as map keys. Assigned to each of the keys there will be a vector loaded with the names of the fields in each table.

5.2.2.2 Sqlite3Db() [2/2]

```
handler::Sqlite3Db::Sqlite3Db (
            std::string db_path )
```

Constructor for user defined database name.

Parameters

db_name	name of the database to be connected to.

```
#include <handler.hpp>
int main(int argc, char const *argv[]) {
  hanlder::Sqlite3Db MyHandler("dbname.db");
  return 0;
}
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

5.2.2.3 ~Sqlite3Db()

```
handler::Sqlite3Db::~Sqlite3Db ( )
```

Destructor of the class Sqlite3Db.

This desctructor will close the connection to the database and then destroy the object.

5.2.3 Member Function Documentation

5.2.3.1 callback()

```
int handler::Sqlite3Db::callback (
            void * NotUsed,
            int argc,
            char ** argv,
            char ** azColName )  [static]
```

Callback to show the output of sqlite3_exec()

When not using the executeQuery() method, it may be interesting to use the sqlite3_exec() instead, and so, a callback to see the output of the operation is higly recommended.

Parameters

NotUsed	Not used.
argc	Number of arguments.
argv	Values of the arguments.
azColName	Name of the column in the database.

5.2.3.2 closeConnection()

```
void handler::Sqlite3Db::closeConnection ( )
```

Close connection to the current database.

The closing of the connection is not recommended, since the main purpose of the Handler is to manage an open database while connected to it.

```
#include <handler.hpp>
int main(int argc, char const *argv[]) {
  hanlder::Sqlite3Db MyHandler("dbname.db");
  /*
  ..............
  operations on database
  .......
   */
  MyHandler.closeConnection();
  return 0;
}
```

5.2.3.3 connectDb()

```
bool handler::Sqlite3Db::connectDb ( )
```

Reconnects the handler to it's linked database.

While doing the reconnection all the data inside of the handler is renewed, in case some changes took place during the time it was offline.

Returns

EXIT_SUCCESS if the db was reopened and it's information loaded correctly. EXIT_FAILURE otherwise.

An example of usage could be as follows:

```
#include <handler.hpp>
int main(int argc, char const *argv[]) {
  hanlder::Sqlite3Db MyHandler("dbname.db");
  /*
  ..............
  operations on database
  .......
   */
  MyHandler.closeConnection();
  ...
  if(MyHandler.reconnectDb() == EXIT_SUCCESS){
    ...
  }
  return 0;
}
```

5.2.3.4 createTable()

```
bool handler::Sqlite3Db::createTable (
            std::string table_name,
            std::vector< FieldDescription > fields )
```

Create a table in the database with the specified parameters.

Creates a table in the database with the provided name and with the descripted fields. Each

Parameters

table_name	Name for the table to be created.
fields	Vector of desriptions, each of them containing name of field and data specifications of the field corresponding field.

Returns

EXIT_SUCCESS if correct. Otherwise EXIT_FAILURE is returned.

An example of usage could be as follows:

```
#include <handler.hpp>
#include <query.hpp>
int main(int argc, char const *argv[]) {
        handler::Sqlite3Db myHandler;
        std::vector<handler::FieldDescription> v =
       {{"ID",query::data::int_+query::data::primary_key+query::data::not_null}, \
                {"AGE", query::data::int_+query::data::not_null}, \
                {"PHONE", query::data::int_+query::data::not_null}, \
                {"NAME",query::data::char_+query::data::len(50)+query::data::not_null}};
        if (myHandler.createTable("COMPANY", v) == EXIT_SUCCESS) {
            ...
        }
        return 0;
}
```

5.2.3.5 deleteRecords()

```
bool handler::Sqlite3Db::deleteRecords (
              std::string table_name,
              std::string condition )
```

Delete the records from a table that meet the provided condition/s.

Locates and deletes the records from the table which meet a condition given. If the condition is "all", all records are deleted.

Parameters

table_name	Name of the table where the records will be deleted.
condition	Condition to match for the deletion.

Returns

EXIT_SUCCESS if correct. Otherwise EXIT_FAILURE is returned.

An example of usage could be as follows:

```
#include <handler.hpp>
int main(int argc, char const *argv[]) {
        handler::Sqlite3Db myHandler;
        //In table company created previously
        std::string table_name = "COMPANY";
        std::string condition = "ID == 1";
        //Delete record that meets condition
        if (myHandler.deleteRecords(table_name, condition) == EXIT_SUCCESS){
            ...
        }
        //Delete all records in the table "table_name"
```

```
        if (myHandler.deleteRecords(table_name, condition="all")){
            ...
        }
        return 0;
}
```

5.2.3.6 dropTable()

```
bool handler::Sqlite3Db::dropTable (
              std::string table_name )
```

Drop the table specified.

Drops the specified table and deletes it's content from the DbInfo contained in the Handler.

Parameters

table_name	Name of the table to be dropped.

Returns

EXIT_SUCCESS if correct. Otherwise EXIT_FAILURE is returned.

An example of usage could be as follows:

```
#include <handler.hpp>
int main(int argc, char const *argv[]) {
        handler::Sqlite3Db myHandler;
        //In table company created previously
        std::string table_name = "COMPANY";
        //Delete record that meets condition
        if(myHandler.dropTable(table_name) == EXIT_SUCCESS){
      ...
    } else{
      (error handling)
      ...
    }
        return 0;
}
```

5.2.3.7 executeQuery()

```
bool handler::Sqlite3Db::executeQuery (
              const char * sql_query,
              std::vector< std::string > & data = empty_vec,
              std::vector< int > indexes_stmt = {},
              bool verbose = false )
```

Execute an SQLite query and receive the output selected.

Parameters

sql_query	The query to be executed.
indexes_stmt	The indexes of the output (each index is a field of the database) that will be extracted from the result of the query.
data	Container to store the data retrieved from the indexes_stmt.
verbose	If set to true, the result of the query will also be printed through the console. Default value is false.

Returns

EXIT_SUCCESS if correct. Otherwise EXIT_FAILURE is returned. The data vector is passed by reference, so the content of it is changed during the query execution.

An example of usage could be as follows:

```
#include <handler.hpp>
#include <query.hpp>
int main(int argc, char const *argv[]) {
  handler::Sqlite3Db MyHandler("mydatabase.db");
  /*
  ...
  */
  /* Generate custom user query */
  const char* query = (query::cmd::select + query::cl::from + "My Table" \
                + query::end_query).c_str();
  /* Vector for data storage */
  std::vector<std::string> data;
  /* Indexes to get the data from in each row of the result */
  std::vector<int> indexes_stmt = {1,3};
  /* Flag for verbose mode */
  bool verbose = true;
  /* Execute the query with the settings defined and extracting data */
  MyHandler.executeQuery(query, data, indexes_stmt, verbose);
  /*
  Process the data
  ....
   */
  /* Generate another custom user query */
  query = (query::cmd::drop_table + "My Table" \
              + query::end_query).c_str();
  /* Execute the query in simple mode, no data extraction and not verbose mode */
  MyHandler.executeQuery(query);
  return 0;
}
```

5.2.3.8 getAffinity()

```
const std::string handler::Sqlite3Db::getAffinity (
             const std::string field_datatype )
```

Calculate the affinity token corresponding to a datatype given.

Using the data affinities defined in the SQLite3 documentation, and with the rules specified to calculate them, this method will return the affinity according to the datatype it receives.

Parameters

field_datatype	The datatype for which the affinity token will be calculated.

Returns

Affinity values "INTEGER", "REAL", "TEXT", "BLOB" or "NUMERIC", depending on the input.

Lambda to convert the input string to uppercase for further processing.

5.2.3.9 getDbPath()

```
std::string handler::Sqlite3Db::getDbPath ( )
```

Gets db relative path from the database.

Returns

A vector containing the names of the tables.

5.2.3.10 getFields()

```
std::vector< std::string > handler::Sqlite3Db::getFields (
            std::string table_name )
```

Get field's names from a table in the database.

Parameters

table_name	The name of the table which field's names are needed.

Returns

A vector containing the names of the fields of the table.

5.2.3.11 getNumTables()

```
int handler::Sqlite3Db::getNumTables ( )
```

Get number of tables in the database.

Returns

The number of tables.

5.2.3.12 getTables()

```
handler::DbTables handler::Sqlite3Db::getTables ( )
```

Get tables information map stored in the handler.

Returns

A vector containing the names of the tables.

5.2.3.13 getTablesNames()

```
std::vector< std::string > handler::Sqlite3Db::getTablesNames ( )
```

Get table's names from the database.

Returns

A vector containing the names of the tables.

5.2.3.14 insertRecord()

```
bool handler::Sqlite3Db::insertRecord (
            std::string table_name,
            std::vector< std::string > values )
```

Insert record data inside of a table.

Given the data to insert, the method will put it inside of the specified table if the fields match.

Parameters

table_name	Name of the table where the record will be added.
values	Container of the values of the fields to insert. The not defined values should contain an empty string ("").

Returns

EXIT_SUCCESS if correct. Otherwise EXIT_FAILURE is returned.

An example of usage could be as follows:

```
#include <handler.hpp>
#include <query.hpp>
int main(int argc, char const *argv[]) {
        handler::Sqlite3Db myHandler;
    //In table company created previously
        std::string table_name = "COMPANY";
                                          //ID  AGE    PHONE      NAME
        std::vector<std::string> values = {"1", "34", "5521664", "James"};
    //Insert record with all fields defined
        if (MyHandler.insertRecord(table_name, values) == EXIT_SUCCESS){
            ...
        }
                                                  //ID  AGE PHONE  NAME
        std::vector<std::string> values_incomplete = {"2", "34", "", "Thomas"};
    //Insert record with one or more fields not specified
        if (MyHandler.insertRecord(table_name, values_incomplete) == EXIT_SUCCESS){
            ...
        }
        return 0;
}
```

5.2.3.15 isAffined()

```
bool handler::Sqlite3Db::isAffined (
            const std::string affinity,
            const std::string value_to_check )
```

Compares the value of some field to it's corresponding affinity.

The value of the data in a field is compared against it's affinity. The comparison determines if the data given is valid for that field or not.

It is important to notice that only "INTEGER", "REAL" and "NUMERIC" affinities need this validation. The "TEXT" or "BLOB" affinities can contain numbers, but numbers cannot contains characters inside of them, except from "," or "." for decimal depending on the country.

Parameters

affinity	The affinity token of the corresponding field to be checked.
value_to_check	The value that needs to be validated before an operation.

Returns

True if the value is valid for the field given, false otherwise.

5.2.3.16 isInt()

```
static bool handler::Sqlite3Db::isInt (
            char c )  [inline], [static]
```

Check if a character is valid for being inside of an integer number.

Parameters

c	The character to be checked.

Returns

True if the character is not alphabetic, a space, or a comma or dot. False otherwise.

5.2.3.17 isReal()

```
static bool handler::Sqlite3Db::isReal (
            char c )  [inline], [static]
```

Check if a character is valid for being inside of a real number.

Parameters

c	The character to be checked.

Returns

True if the character is not alphabetic or a space. False otherwise.

5.2.3.18 isValidInt()

```
bool handler::Sqlite3Db::isValidInt (
            const std::string & str )  [inline]
```

Checks if a string is a valid integer number.

Parameters

str	String to be checked.

Returns

True if the string is a valid integer. False otherwise.

5.2.3.19 isValidReal()

```
bool handler::Sqlite3Db::isValidReal (
            const std::string & str )  [inline]
```

Checks if a string is a valid real number.

Parameters

str	String to be checked.

Returns

True if the string is a valid real (float, double...). False otherwise.

5.2.3.20 selectRecords() [1/2]

```
std::vector< std::string > handler::Sqlite3Db::selectRecords (
            select_query_param select_options )
```

Selects and extracts the records that meet certain conditions.

Parameters

select_options	Structure containing all the necessary options to be used during the select statement.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

/include selectRecordsStruct.cpp Lambda to convert the input string to uppercase for further processing.

5.2.3.21 selectRecords() [2/2]

```
std::vector< std::string > handler::Sqlite3Db::selectRecords (
            std::string table_name,
            std::vector< std::string > fields = {"*"},
            bool select_distinct = false,
            std::string where_cond = "",
            std::vector< std::string > group_by = {},
            std::string having_cond = "",
            std::vector< std::string > order_by = {},
            std::string order_type = "ASC",
            int limit = 0,
            int offset = 0 )
```

Selects and extracts the records that meet certain conditions.

Extracts all the data that fits the descriptions and conditions passed as arguments. Most of them are optional, so only the table name is necessary if no other specific parameters are needed.

Parameters

table_name	Name of the table from where the data will be selected.
fields	Container of the names of the fields of the table that will be retrieved from the results in string format. Default value is "*" to take all the fields.
select_distinct	Boolean flag to set whether or not only unique results should be selected. Default value is false.
where_cond	If set, it contains the condition to apply with a WHERE clause in the select query composition. It is empty by default.
group_by	If set, it contains the condition that will be used to group the results of the select query. It is empty by default.
having_cond	If set, it contains the condition applied to the query after a HAVING clause. It is empty by default.
order_by	If set, it contains the condition that will be used to order the results of the select query. It is empty by default.
order_type	Only applied if the order_by argument is set. Defines the type of ordering to be applied. The types are "ASC" or "DESC". Default value is "ASC"
limit	If set, it defines the number of results that will be extracted from the select query data. Default value is 0 for no limit.
offset	If set, it defines the number of results that will be skipped from the select query data before extracting them. Default value is 0 for none.

Returns

A vector containing all the values retrieved in order. This means that, if three fuelds were given as argument, each group of three elements of this vector will correspond to a row of data.

/include selectRecords.cpp Lambda to convert the input string to uppercase for further processing.

5.2.3.22 updateHandler()

```
bool handler::Sqlite3Db::updateHandler ( )
```

Updates the information contained in the handler.

If multiple connections are being used in the same database, it may be possible that a table is changed, or a new one created, without the current hanlder knowing it. For this purpose, execution of this method will updated all the tables and field information that is stored in the handler, so it can operate normally from different connections.

Returns

EXIT_SUCCESS if the information was updated. EXIT_FAILURE if an error occurred during the process.

An example of usage could be as follows, with two different programs working on same db:

```
#include <handler.hpp>
#include <query.hpp>
int main(int argc, char const *argv[]) {
        /* Stablish first connection to database */
        handler::Sqlite3Db MyHandler("mydatabase.db");
    while (/* condition */) {
      /* update every time before acting on the db in case something is changed */
      MyHandler.updateHandler();
      /*
           ....
           operations on db.
           ...
       */
    }
        return 0;
}
#include <sqlite3handler.hpp>
#include <sqlite3query.hpp>
int main(int argc, char const *argv[]) {
        /* Stablish second connection to the database */
        handler::Sqlite3Db AnotherHandler("mydatabase.db");
    while (/* condition */) {
      /* update every time before acting on the db in case something is changed */
      AnotherHandler.updateHandler();
      /*
           ....
           operations on db.
           ...
       */
    }
        return 0;
}
```

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

- include/handler.hpp
- src/sqlite3handler.cpp