Database mapped storage

Explanatory notes

Ilia Udalov
Mikhail Kuprianov

Prepared for
Abvanced Database Seminar
Scientific supervisor: Dudarev V.A
16.12.2015

Contents

1. Task statement.

The project “Database mapped storage” is a C++ library that provides the container for storing, modifying and accessing large amount of data (more then RAM available), relational databases used as storage.

Available database management systems:

- SQLite;
- PostgreSQL;
- third party database support can be easily added.

2. Group members. Responsibility.

Ilia Udalov:

- Interaction with external databases API.
- DB Wrappers;
- Generic architecture;
- PostgreSQL server management.

Mikhail Kuprianov:

- C++ interfaces;
- Dynamic queries generation;
- Generic architecture;
- Performance tests.

3. Library functions and features:

3.1. Building from sources

Library designed for POSIX like Operating System, can be compiled by GCC or Clang compilers (that provides C++11 extension). The library is not self-contained and requires PostgreSQL client library and headers (libpq). SQLite sources are built-in. So SQLite storage can be used as “standalone” library without external dependencies. Library can by build into shared library file (.so or .dylib) by command: `./gradlew shared`

3.2. Application programming interface (API)

The library API is below.

<u>Template class Datapack:</u>

Container for storing data.

`Datapack(std::string _name, DBWrap& _db)` – obligatory constructor, accepts name of object and database for string;
`void Push(const Args&...args)` – add data for container;
`void Request(const Query& rule = "")` – set constrains for data;
`void Remove(const Query& rule = "")` – remove data from container;
bool Get(Args*...args) – get data entries.

<u>Class ExecResult:</u>

Result of select statent (table representation).
`std::vector<std::string> operator[](size_t row)` – get row at index;
`size_t cols()` – get number of columns;
`size_t rows()` – get number of rows;

<u>Interface DBWrap:</u>

Defines interface for interacting with databases inside library. Using this interface user can provide support for other database management system.
`virtual void touch(const std::string& connectionString)` – connect to database using connectionString as parameter.
`virtual ExecResult execute(const std::string& query)` – execute given query;
`virtual void close()` – close database connection.

<u>Class SQLiteWrap:</u>

Wrapper for SQLite database, implements DBWrap interface, does not provide any extra functionality.

<u>Class PostgreSQLWrap:</u>

Wrapper for PostgreSQL database, implements DBWrap interface, does not provide any extra functionality.

4. Benchmarks

SQLite

10 000 adding operations(INSERT)

- AVG: 0.02s
- Average: 115.4s

3 type of reqeuest(SELECT)

- simple request(SELECT *)
    - 0.016176
- simple request that return about half of data(random fields)
    - 0.006836
- huge request with 100+ conditions
    - 0.011508
- string “like” operation
    - 0.011865

PostgreSQL

10 000 adding operations(INSERT)

- AVG: 0.0158s
- Average: 62s

3 type of reqeuest(SELECT)

- simple request(SELECT *)
    - 0.0335
- simple request that return about half of data(random fields)
    - 0.009336
- huge request with 100+ conditions
    - 0.051508

5. Conclusion.

Result of our project is open source POSIX compatible library for working with big data sets. All docs, sources, test and etc. available at github.com/iudalov/DBMS. Main features are:

- Portability;
- Platform independent storage;
- Concurrent access;
- Speed;
- Extendibility.

The library supports several database management systems foe storing data:
SQLite;
PostgreSQL;

Also there is ability to extend library by adding support of extra database management systems using standard API.

Performance tests were conducted for two database management systems (PostgreSQL and SQLite) related to current library implementation.

Possible future enchantments:

- Smart batch processing with auto data loading
- Extra database management systems support
- Windows OS support(Windows DBMS as well)
- Extended data types support
- Database performance optimization

6. Literature references.

1. https://github.com/iudalov/DBMS
2. http://sqlite.org/
3. http://postgresql.org/
4. http://cppreference.com/
5. http://habrahabr.ru/post/101430/

7. Appendixes.

1 Usage example

```
// declaring data
struct Person
{
    std::string firstName;
    std::string lastName;
    int         age;
    void log() {
        printf("firstName: %s\nlastName: %s\nage: %d\n",
firstName, lastName, age);
    }
};

// sample data
const Person sample_data[] = {
        {"Chandler", "Bing", 25},
        {"Monica", "Geller", 24},
        {"Ross", "Geller", 27},
        {"Fibi", "Bufe", 24},
        {"Joey", "Tribbiani", 25},
        {"Pachel", "Green", 23},
    };


void main() {
    // creating databse wrapper object
    std::shared_ptr<DBWrap> db =
std::make_shared<PostgreSQLWrap>();

    // connectiong to datablase
    db->touch("user=johndoe host=example.com
dbname=containerstmp");

    // creating container
    Datapack<std::string, std::string, std::string, int, double>
pack("Friends", *db);

    // filling container
    for(auto& i: sample_data)
    {
        pack.Push(i.fname, i.mname, i.lname, i.age, i.height);
    }

    // providing constrains
    pack.Request(pack[1] % "%Geller%");

    // getting results
    Person p;
    while(pack.Get(&p.fname, &p.mname, &p.lname, &p.age,
&p.height)) {
        p.log();
    }
}
```

2 Wrapper interfaces

```
class DBWrap {
public:
    virtual ~DBWrap(){}
    virtual std::string version() = 0;
    virtual void touch(const std::string& connectionString) = 0;
    virtual ExecResult execute(const std::string& query) = 0;
    virtual void close() = 0;
};
```