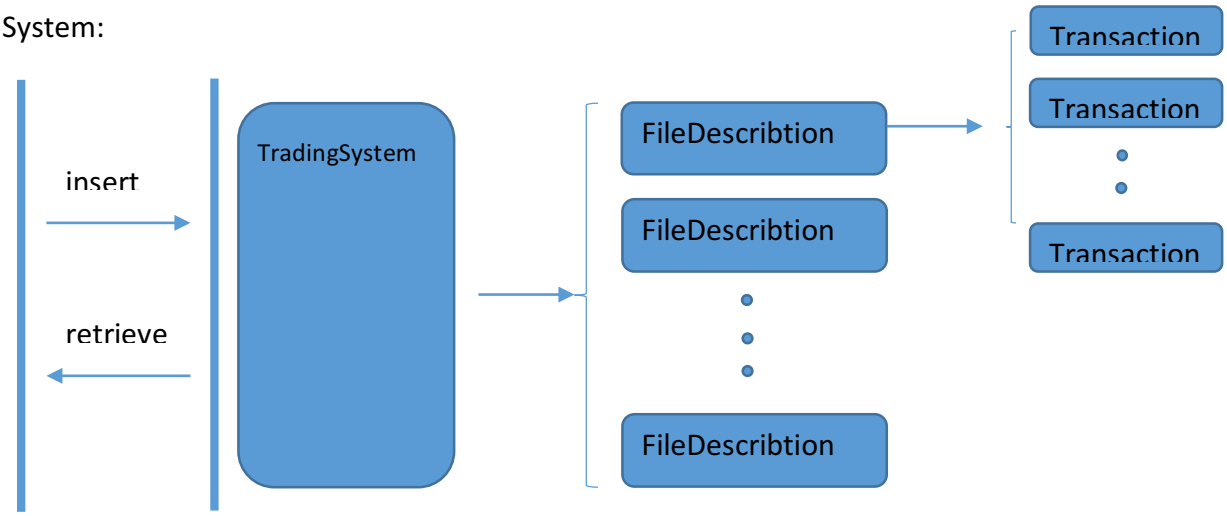
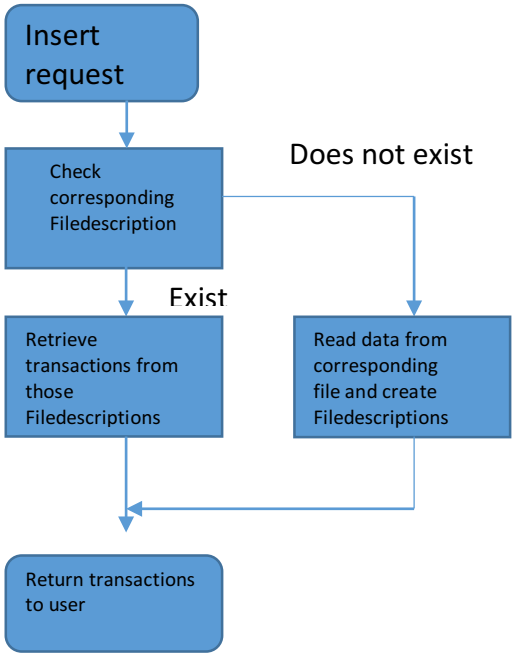
Unit System:



How to store data?
Data will be distributed by date.
They will be stored as file like 13_05_2017.dat, 14_05_2017.dat
In memory, the data will be present as class Transaction. Class FileDescription used to manage transactions for specific file.

In FileDescription, transaction will be sorted by time.
If most transactions will be send as chronological order, sorted vector or list should be fine.
Otherwise, we might need consider Red-Black-Tree for better insertion efficiency while it will consume more memory.



Since FileDescription managed transactions sorted, we can find transaction for specify time in log(n) Time complexity.

TradingSystem can manage specific number of FileDescription, which depend on system memory. When TradingSystem find the corresponding FileDescription does not exist in memory. It will write some FileDescription into file and release the memory for new FileDescription.
To reduce the read / write operation, the system use LFU(Least Frequently Used) cache algorithm to manage FileDescriptions.

LFU:
Three unordered_map used to implement this LFU algorithm.
Retrieve FileDescription by data: Time complexity O(1), Space Complexity O(k)
Create new FileDescription by data: Time complexity O(1), Space Complexity O(k)
k is the capability of this LFU algorithm, which might vary from different system.

Retrieve:

1. Parsing date and collecting corresponding FileDescriptions.
2. If some FileDescriptions do not exist, use LFU algorithm to create them.
3. Retrieve data by date with binary search or specific search based on Red-Black-Tree.
4. Return data to user.

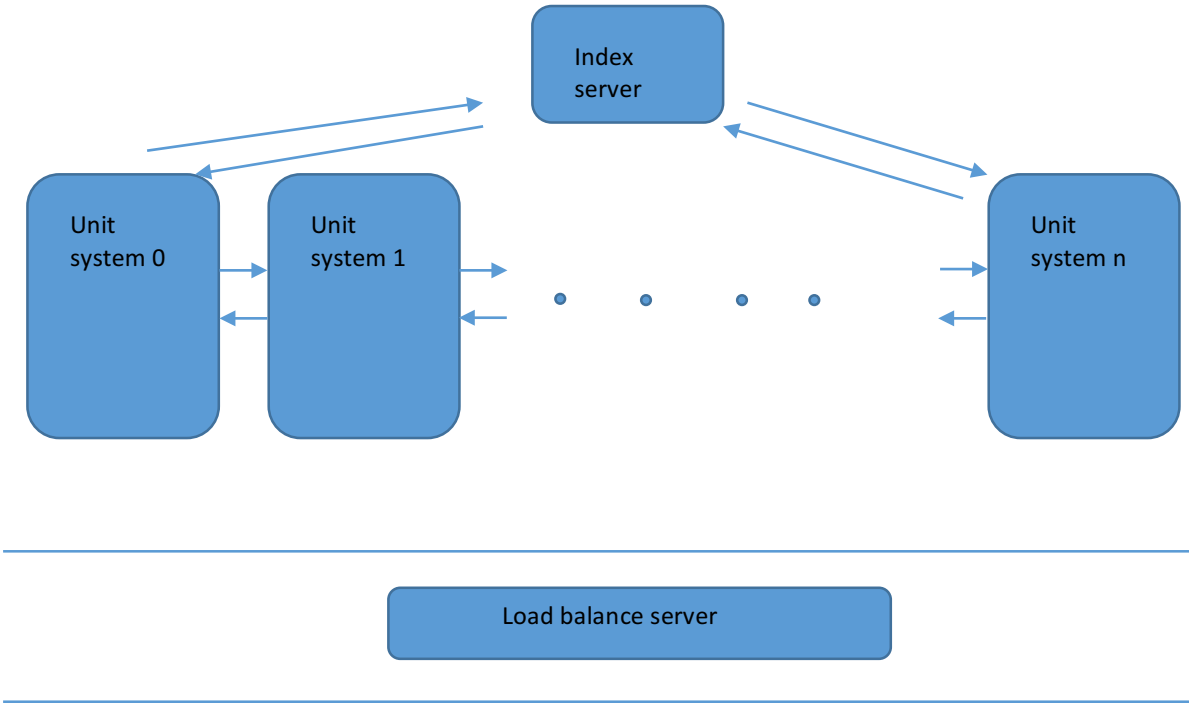
Simple class design:

```
class TradingSystem {
public:
    TradingSystem() {}
    bool InsertTransaction(Transaction& transaction);
    // time format: dd__mm__yyyy-HH:MM:SS. Time_begin == time_end means retreieve data at
//specific time
    std::vector<Transaction> RetrieveTransactions(std::string time_begin, std::string time_end);
private:

    std::shared_ptr<FileDescription> GetFileDescriptionByDate(std::string data);
    std::shared_ptr<FileDescription> CreateFileDescriptionByDate(std::string data);


    int min_frequent_ = 0, capacity_ = TRADINGSYSTEM_FILEDESCRIPTION_CAPACITY;

    // key, {FileDescription, freq}
    std::unordered_map<std::string, std::pair<std::shared_ptr<FileDescription>, int>>
mp_file_description_;
    // key, iterator of
    std::unordered_map<std::string, std::list<std::string>::iterator> mp_iterator_;
    // freq, list of key
    std::unordered_map<int, std::list<std::string>> mp_list_of_freq_;
};
```

```
class FileDescription {
public:
    FileDescription(std::string date);

    bool InsertTransaction(Transaction& transaction);
    void RetrieveTransactionsBetweenTime(std::vector<Transaction> transactions, std::string
time_start, std::string time_end);
    void WriteToFile();
private:
    std::string date_;
    std::vector<Transaction> transactions_;
};


class Transaction {
public:
    Transaction(time_t date_time, std::string stock_symble, double price) :
        date_time_(date_time), stock_symble_(stock_symble), price_(price) {}

    std::string GetStringDate() {
        return DateOperation::GetStringDate(date_time_);
    }
    time_t GetDateTime() {
        return date_time_;
    }
    std::string GetStockSymble() {
        return stock_symble_;
    }
    double GetPrice() {
        return price_;
    }
private:
    time_t date_time_;
    std::string stock_symble_;
    double price_;
    //
    // some additional information
    //
};
```

Distributed System

Basic design



Retrieve:

1. Users send request to load balance server and receve ip of indicated unit system
2. Userr send request to indicated unit system.
3. Indicated Unit system will retrieve local data with the same algorithm as an single unit system.
4. Unit system will query index server for other unit systems who might contain related data.
5. Unit system send request to other unit system based on the result from index server.
6. After unit system collected data from other unit system and local data, it will send the result to user.

Insert:

1. Users send request to load balance server and receve ip of indicated unit system
2. Userr send request to indicated unit system.

3. Indicated unit system will create record with the same algorithm as an single unit system and update information on index server if need.

Optionnal optimizer
Unit system does not response for collecting data from other possible unit systems. indicated unit system will send user location data and ip address of other possible unit systems. User need collect other data from those server with local data request(send local data only).