Документация на MiniDB

Анализ на задачата и подход за решение

Целта на проекта е създаване на **конзолно приложение за работа с бази от данни**, с възможност за:

- Създаване, редактиране и изтриване на таблици
- Добавяне на колони с конкретен тип
- Вмъкване на редове с различни стойности (int, double, string, date, NULL)
- Търсене, селектиране, обновяване и изтриване на данни
- Запазване/зареждане на базата в/от файл (CSV формат)

Подход

Проектът беше разделен на следните стъпки:

1. **Абстракция на данните** чрез `Cell` клас и негови наследници (`IntCell`, `StringCell`, и др.)
2. **Създаване на класовете** `Column`, `Row`, `Table`, и `Database`
3. Имплементиране на **динамична типизация** с помощта на виртуални функции
4. Сигурно управление на паметта и копиране чрез `clone()`
5. **Команден интерфейс** за изпълнение на команди
6. **Файлова обработка** с експортиране в CSV и зареждане от файлове

Решени проблеми

- Сигурно клониране на полиморфни обекти с `Cell* clone()`
- Обработка на `NULL` стойности без крашове
- Валидиране на входове за типове (вкл. дати и кавичките при стрингове)
- Избягване на memory leaks с помощта на `std::vector<Cell*>` и `freeDynamic`

Класове и архитектура

`class Cell` *(абстрактен клас)*

- Представлява клетка с виртуални методи:
  - `clone()`, `toString()`, `equals()`, `getType()`

Наследници:

- `IntCell`, `DoubleCell`, `StringCell`, `DateCell`, `NullCell` – всеки съхранява конкретна стойност и имплементира нужната логика.

`class CellFactory`

Отговаря за създаване на обекти от тип `Cell*` според подаден низ и тип на колоната.

- **Методи:**
  - `Cell* createCellFromString()` – Анализира `value` и създава подходящ обект: `IntCell`, `DoubleCell`, `DateCell`, `StringCell`, `NullCell`
  - `std::string parseEscapedString()` – Декодира escape-нати символи като `\"Ivan\"` → `Ivan`
  - `bool parseDateString()` – Проверява дали даден стринг е валидна дата в `YYYY-MM-DD` формат

`class Date`

Представлява проста структура за дата.
Използва се в `DateCell`.

- **Член-данни:**
  `int day, month, year;`
- **Методи:**
  - `toString()` – връща датата във формат `YYYY-MM-DD`
  - `operator==(); operator!=()` – сравнение между две дати

`class Column`

Роля: Представлява колона в таблица – съдържа име и тип на данните.

- **Член-данни:**
  - `std::string name` – името на колоната
  - `std::string type` – типът на данните
- **Методи:**
  - `Column(const std::string& name, const std::string& type)` – Създава валидирана колона
  - `const std::string& getName() const` – Връща името
  - `const std::string& getType() const` – Връща типа
  - `void setName(const std::string&)` – Валидира и присвоява ново име
  - `void setType(const std::string&)` – Валидира и задава нов тип (от позволените)

Забележка: Валидиращите методи хвърлят изключения при невалиден тип.

`class Row`

Роля: Представлява ред в таблицата – съдържа вектор от указатели към клетки (`Cell*`).

- **Член-данни:**
  - `std::vector<Cell*> cells`
- **Методи:**
  - `Row(const std::vector<Cell*>&)` – Създава ред с копирани клетки (чрез `clone`)
  - `Row(const Row&), Row& operator=(const Row&)` – Дълбоко копиране
  - `~Row()` – Освобождава динамичната памет
  - `std::vector<Cell*>& getCells()` – Достъп до клетките
  - `std::string toString() const` – Форматира реда като CSV низ

Забележка: Всеки `Cell*` в `Row` се клонира при копиране, за да се избегне споделена памет.

`class Table`

Роля: Представлява една таблица – съдържа редове и колони, и предоставя пълна функционалност за работа с тях.

- **Член-данни:**
  - `std::string name` – името на таблицата
  - `std::vector<Column> columns` – колоните
  - `std::vector<Row> rows` – редовете
- **Методи:**
  - `Table(const std::string&)` – Създава таблица с име
  - `void addColumn(const std::string& name, const std::string& type)` – Добавя нова колона и допълва всеки ред с `NullCell`
  - `void insertRowFromStrings(const std::vector<std::string>&)` – Създава клетки от низове и добавя ред
  - `void deleteRows(size_t columnIndex, const std::string& value)` – Изтрива редове по стойност
  - `void update(size_t searchCol, const std::string& searchVal,size_t targetCol, const std::string& newVal)` –Ппроменя стойност в ред при съвпадение
  - `void select(size_t columnIndex, const std::string& value) const` – Извежда редове, съответстващи на стойност
  - `void print() const` – Отпечатва цялата таблица
  - `void describe() const` – Отпечатва колоните и типовете
  - `void exportToFile(const std::string&) const` – Записва таблицата във `.csv`

- `const std::vector<Row>& getRows() const;`
  `std::vector<Row>& getRows()` – Достъп до редовете

`class Database`

Роля: Представлява цялата база от данни – управлява множество таблици и отговаря за зареждане, записване и навигация.

- **Член-данни:**
  - `std::vector<Table> tables` – всички таблици в базата
- **Методи:**
  - `void addTable(const Table&)` – Добавя таблица (с проверка за уникалност на името)
  - `Table* getTableByName(const std::string&)` – Намира таблица по име
  - `void describe(const std::string&)` – Описва колоните в таблица
  - `void showTables() const` – Показва имената на всички таблици
  - `void save(const std::string& filename) const` – Записва всички таблици във файл
  - `void load(const std::string& filename)` – Зарежда всички таблици от файл
  - `void clear()` – Изчиства всички таблици от паметта

Забележка за паметта

- Всички клетки се съхраняват като `Cell*`, но се управляват чрез:
  - `clone()` метод при копиране на редове
  - `freeDynamic()` метод при освобождаване на памет

Идеи за бъдещи подобрения

- **Използване на `smart pointers` (`std::unique_ptr`)** за управление на памет вместо raw `Cell*`
- **Пълна поддръжка на SQL-подобен синтаксис** (пример: `SELECT * FROM users WHERE id = 5`)
- **Поддръжка на тип `bool` или `enum` колони**
- **Имплементиране на сортиране по колона**
- **Система за индексиране** за по-бързи търсения