***Бази от данни***

*Ния Добринова Митева, ФН: 0MI0800438, I курс, 3 група*

*Github repo: https://github.com/NiaDMiteva/Command-Line-Database-Management*

***Описание и идея на проекта:***

Основната идея на проекта е да се реализира локална база от данни, която да може да бъде управлявана през командния ред. Информацията за данните в базата трябва да се съхранява във файлове, като има един основен, който пази имената на таблиците в базата и имената на файловете, в които е съхранена тяхната информация.

***Цел и задачи на разработката:***

Целта на проекта е да се реализира пълната функционалност, необходима за правилната и лесна работа с базата данни в диалогов режим. Основните задачи включват:

1. Изграждане на структура – проектът трябва да бъде организиран по начин, който следва добрите практики в обектно-ориентираното програмиране (ООП), което включва правилното разделение на отговорностите между различните класове и компоненти.
2. Създаване на ефективен формат за съхранение на данни – информацията трябва да бъде съхранена във файлове по начин, който да осигурява бърз и лесен достъп до данните, като същевременно гарантира целостта и сигурността на информацията.
3. Изграждане на добри връзки между файловете – връзките между различните файлове, които съхраняват данни за таблиците и колоните, трябва да бъдат добре дефинирани и надеждни, за да се осигури консистентност на данните в базата.
4. Реализиране на удобен потребителски интерфейс – командният ред трябва да бъде лесен за използване, с интуитивни команди, които позволяват бързо и ефективно управление на базата данни. Това включва възможности за добавяне, редактиране, изтриване и извличане на данни, както и управление на самите таблици.
5. Създаване на механизъм за възстановяване – по време на работа на програмата трябва да се създават временни файлове за възстановяване, които да пазят незаписаните промени в случай на ненавременно прекратяване на програмата. Това ще осигури възможност за възстановяване на последните промени и ще предотврати загубата на данни.
6. Имплементиране на необходимите функции – всички функции, които може да притежава една база данни, трябва да бъдат реализирани по такъв начин, че те да могат да се извикват лесно и интуитивно през командния ред. Това включва основни операции като създаване на таблици, добавяне на записи, обновяване на данни, изтриване на записи и извличане на информация.

***Обща архитектура:***

Както бе споменато по-горе, основната задача при изграждането на архитектурата е спазването на ООП принципите. Проектът е разделен на класове, като всеки клас отговаря за конкретна част от задачата. Един основен клас разпределя работата между останалите. За всеки клас е приложен принципът *RAII* (*Resource Acquisition Is Initialization*). Използвани са няколко шаблони за дизайн: *Factory, Prototype* и *Singleton*.

Основната архитектура включва абстрактен клас *Column* с наследници - *IntColumn, DoubleColumn* и *StringColumn*. Причината те да са наследници, а не шаблонни класове, е че имат различна логика спрямо типа на данните за метода *validateValue*. *IntColumn* и *DoubleColumn* имат специфичен метод – *parseValue*. Класът *Table* съдържа хетерогенен контейнер от колони, а класът *Database* съдържа контейнер от таблици, които управляват колоните и самите таблици в базата данни.

Създаден е и помощен клас *StringUtils* за обработка на стрингове, който е статичен и се използва като *namespace*. Реализиран е интерфейс *Serializeable*, наследен от няколко други класа за запис и четене на обекти от файл. За реализацията на хетерогенния контейнер от колони е използван *Prototype Design Pattern*, което позволява създаването на колони по подаден тип. *Factory Design Pattern* е използван за функцията *createColumn*, която създава конкретен тип колона според подадената информация.

***Реализация:***

Програмата съдържа три основни абстракции: колона, таблица и база данни:

1. *Column* – Абстрактен клас с три наследника: *IntColumn, DoubleColumn и StringColumn*. Всяка колона съдържа вектор от стойности и извършва определени операции върху тях, като добавяне, премахване и валидиране на стойности. Колоните наследяват *Serializeable*, за да записват информацията си във файловете на таблиците, които ги съдържат.
2. *Table* – Съдържа хетерогенен контейнер от колони и изпълнява операции върху тях. Освен основните операции като добавяне и изтриване на колони, таблицата има методи за филтриране и агрегиране на данни. Всеки *Table* обект има определен файл за записване на информацията си и логика за създаване на файл за възстановяване. Поради необходимостта от хетерогенен контейнер, класът *Table* не наследява *Serializeable*, но съдържа функции за сериализация и десериализация. Таблицата има и приятелска функция за извършване на *Inner Join* операция между две таблици, която в последствие не води до правилен резултат спрямо текущата реализация.
3. *Database* – Съдържа контейнер от уникални таблици и предоставя методи за управление на тези таблици. Класът *Database* наследява *Serializeable* и има метод за създаване на файл за възстановяване, който съхранява състоянието на базата данни в случай на аварийно прекъсване на програмата. Класът включва и методи за добавяне, премахване и търсене на таблици, както и за изпълнение на сложни заявки върху данните.
4. Класът *DatabaseService* се грижи за връзката между потребителя и програмата, обработвайки подадените команди и управлявайки файловете с информацията за базата данни. Той е *Singleton*, понеже се инстанцира само веднъж във функцията *main*, която извиква метода *run* на тази инстанция. Освен това, този клас е отговорен за изписването на конзолата и обработката на грешки, винаги успявайки да завърши.
5. Класът *StringUtils* е статичен и служи като *namespace* за функции, обработващи стрингове. Той се използва главно за обработка на команди, въведени от потребителя, и предоставя помощни методи за манипулация на текстови данни.

***Обобщение на изпълнението на началните цели:***

Всички функционалности, изисквани от проекта, са реализирани, без innerJoin. Програмата обработва командите на потребителя интуитивно, а класовете работят добре помежду си, което означава, че началните цели на проекта са постигнати. Архитектурата на файловете е изградена добре, като те се съхраняват в подходящ формат, който работи оптимално с програмата. Програмата обработва грешките така, че винаги е в коректно състояние, успява да се възстанови от грешки и предоставя необходимите съобщения на потребителя.

***Насоки за бъдещо развитие и усъвършенстване:***

Първо, бих реализирала по-универсален файлов формат като JSON или XML, за да направя файловете по-лесни за четене и обработка от други програми. Това би улеснило интеграцията с други приложения и системи, които използват тези формати. Бих създала графичен потребителски интерфейс, който да замени командния ред, правейки приложението по-удобно за обикновените потребители и премахвайки нуждата от излишни валидации за коректност. Това би увеличило потребителската база и би направило програмата по-достъпна за по-широк кръг от потребители, които не са запознати с командния ред.