پروژه UniDB

طراحی موتور دیتابیس NoSQL

فاز اول

ساختمان‌داده‌ها

دکتر افسانه فاطمی

پاییز ۱۴۰۴

تاریخ تحویل:

۱۴۰۴/۰۹/۲۱

📘 مباحث تحت پوشش

آرایه‌ها، لیست‌های پیوندی، پشته، صف، تحلیل پیچیدگی، معماری لایه‌ای

🧭 مقدمه

در این پروژه، شما هسته مرکزی یک دیتابیس NoSQL شبیه MongoDB را پیاده‌سازی می‌کنید.
تمرکز اصلی پروژه علاوه بر تمرین ساختمان داده‌ها، رعایت **معماری تمیز (Clean Architecture)** است.
کد شما باید ماژولار باشد و از کدنویسی اسپاگتی جلوگیری شود.

دیتابیس طراحی‌شده با نام **UniDB** داده‌ها را در قالب **سند (Document)** ذخیره می‌کند.

🏗️ معماری سیستم (System Architecture)

سیستم شامل سه لایه اصلی است و جریان داده از بالا به پایین حرکت می‌کند.

🔍 ۱. لایه تحلیل (Query Parser)

- **ورودی:** رشته متنی خام مانند

"db.students.insertOne(...)"

- **وظیفه:**
  ✔ توکن‌سازی
  ✔ تشخیص نوع دستور
  ✔ استخراج پارامترها

- **خروجی:** ایجاد آبجکت Command

⚙️ ۲. لایه اجرا (Execution Engine)

- **وظیفه:** اجرای دستورات بر اساس شیء Command

- **مدیریت:**
  - صف دستورات (Queue)
  - پشته تراکنش‌ها (Stack)

- ارسال دستورات به لایه ذخیره‌سازی (Storage Engine)

💾 ۳. لایه ذخیره‌سازی (Storage Engine)

- مدیریت مستقیم داده‌ها در RAM
- بدون وابستگی به متن کوئری یا پارسر
- ارائه عملیات‌های پایه:
  add, delete, get

📂 ساختار پیشنهادی پروژه (File Structure)

```
src/
├── main.cpp / Main.java          (Entry Point)
├── models/
│   └── Student.h                 (Data Class)
├── parser/
│   └── QueryParser.h             (Layer 1)
├── engine/
│   ├── ExecutionEngine.h         (Layer 2)
│   ├── Command.h
│   └── TransactionStack.h
└── storage/                      (Layer 3)
    ├── Collection.h              (Interface)
    ├── ArrayCollection.h
    └── LinkedListCollection.h
```

⚠️ **توجه:** تحویل همه کدها در یک فایل باعث کسر نمره خواهد شد.

🧱 نیازمندی‌های پیاده‌سازی

💾 لایه ذخیره‌سازی (Storage Engine)

📌 کلاس Collection (اینترفیس)

باید دو نسخه از آن پیاده‌سازی شود:

📚 ۱. ArrayCollection

- استفاده از **آرایه پویا (Dynamic Array)**
- عملکردها:

➤ insertOne → **O(1)**
➤ deleteOne → **O(n)**
➤ findByID → **O(n)**
➤ findAll
➤ توابع تجمیعی (count, sum, average)

🔗 ۲. LinkedListCollection

- پیاده‌سازی **لیست پیوندی دوطرفه**
- عملیات با تغییر پوینتر انجام می‌شود
- جستجو خطی است
- توابع مشابه نسخه آرایه‌ای

🔎 توابع جستجو

- findAll()
- filter(field, value)

⚙️ لایه اجرا (Execution Engine)

🧱 ۱. مدیریت تراکنش‌ها (Stack)

- beginTransaction()
- rollback() (بازگشت با پاپ‌کردن پشته)
- commit() (تثبیت)

📨 ۲. مدیریت صف دستورات (Queue)

- batch.start → ورود به حالت دسته‌ای
- batch.execute → اجرای ترتیبی صف

🧮 لایه تحلیل (Query Parser)

- استفاده از روش‌های ساده رشته‌ای
- تبدیل دستور به شیء Command
- نیازی به کامپایلر پیچیده نیست

📥 بارگذاری انبوه داده‌ها (Bulk Load)

db.students.import("filename.csv")

- خواندن فایل CSV خط‌به‌خط
- تبدیل رکورد به سند
- درج سریع
- فرمت: id,name,gpa

👨‍💻 نحوه تعامل برنامه (Client Simulation)

🔹 عملیات پایه

```
db.students.insertOne({_id: 101, name: "Ali", gpa: 18.5})

db.students.findByID(101)

db.students.findAll()

db.students.deleteOne({_id: 101})

db.students.import("data_50k.csv")
```

🔹 جستجو و Aggregation

```
db.students.filter("name", "Ali")

db.students.count()

db.students.sum("gpa")

db.students.average("gpa")
```

تراکنش‌ها

```
db.beginTransaction()

db.students.insertOne({ ... })

db.students.deleteOne({ ... })

db.rollback()

db.commit()
```

Batch Processing

```
db.batch.start()

db.students.insertOne(...)

db.students.deleteOne(...)

db.batch.execute()
```

📊 بخش تحلیلی (Report)

🧩 ۱. جدول پیچیدگی زمانی (Big O)

عملیات	Array Collection	LinkedList Collection	دلیل تفاوت
insertOne (آخر)	؟	؟	؟
deleteOne (اول)	؟	؟	؟
deleteOne (وسط)	؟	؟	؟
findByID	؟	؟	؟
count	؟	؟	آیا می‌توان آن را O(1) کرد؟

⏱️ ۲. بنچ‌مارک (Benchmark)

با ۵۰٬۰۰۰ رکورد:

- حذف ۵۰۰ رکورد از ابتدای لیست
- حذف ۵۰۰ رکورد از انتهای لیست
- جستجوی ۵۰۰ رکورد تصادفی

⚠️ **توجه:** نتایج باید روی نمودار رسم شود.

❓ ۳. سؤالات مفهومی

- چرا در لیست پیوندی، deleteOne(id) زمان‌بر است؟
- چرا جدا بودن Parser از Storage مفید است؟ (مثلاً برای افزودن SQL)

🏆 بخش امتیازی پروژه

🔥 سطح ۱: دیتابیس تحت شبکه
(Networked DB – امتیاز: ۲۰۰)

در این سطح، برنامه باید از حالت Shell کنسولی خارج شده و به یک TCP Server واقعی تبدیل شود.

✨ **الزامات:**

- سرور باید روی یک پورت مشخص مثل 8080 گوش دهد.
- تمامی دستورات دیتابیس از طریق سوکت شبکه دریافت شوند.
- یک کلاینت ساده بنویسید که:
  - به سرور وصل شود
  - دستورات را ارسال کند
  - خروجی را دریافت و چاپ کند
- چالش اصلی:
✔️ سرور باید بتواند همزمان به چند کلاینت پاسخ دهد (با استفاده از Multi-threading یا Thread Pool)

💾 سطح ۲: ذخیره‌سازی دائمی

(File Persistence — امتیاز: ۲۰۰)

در این سطح، داده‌ها باید از حالت موقت خارج شده و امکان ذخیره‌سازی روی دیسک فراهم شود.

✨ الزامات:

- با اجرای دستور commit() باید:
  - کل داده‌های داخل حافظه سریالایز شده
  - در یک فایل (مثلاً db.unidb) ذخیره شوند
- با اجرای دوباره برنامه، باید:
  - فایل ذخیره‌شده خوانده شود
  - تمام داده‌ها دوباره داخل حافظه بارگذاری شوند
- می‌توانید از فرمت‌های ساده مثل JSON یا باینری استفاده کنید

🗂️ سطح ۳: فایل‌سیستم پیشرفته

(Random Access IO — امتیاز: ۲۵۰)

این مرحله مخصوص افراد حرفه‌ای و علاقه‌مند به طراحی دیتابیس واقعی است.

✨ **الزامات:**

- دیگر نباید کل فایل را در هر commit بازنویسی کنید
- باید از تکنیک‌های Random Access IO استفاده کنید:
  - `fseek` در C/C++
  - `RandomAccessFile` در Java
- هنگام تغییر رکورد:
  - باید فقط همان چند بایت مربوط به رکورد تغییر کند
- هنگام حذف رکورد:
  - مقدار رکورد باید به عنوان Free Space یا Hole علامت‌گذاری شود
  - رکوردهای بعدی می‌توانند روی همان بلاک نوشته شوند

این بخش شبیه‌سازی ساده‌ای از Page Management در دیتابیس‌های واقعی است.

⚠️ **توجه: فقط در صورتی امتیاز بخش‌های امتیازی را دریافت می‌کنید که حداقل ۹۰٪ از امتیاز بخش اصلی پروژه را کسب کرده باشید.**
در غیر این صورت، حتی اگر بخش امتیازی را کامل پیاده‌سازی کنید، هیچ امتیاز اضافه‌ای برای شما لحاظ نخواهد شد.

🧼 قوانین کلین‌کد (Clean Code Rules)

(کسر نمره تا ۲۰۰- نمره)

این بخش اجباری است و رعایت نکردن آن باعث کسر شدید نمره می‌شود.
هدف این است که پروژه از یک کد تمرینی ساده تبدیل شود به کدی قابل نگه‌داری، معماری‌مند، و حرفه‌ای.

✨ قوانین کلیدی:

✅ ۱. معماری لایه‌ای باید کاملاً رعایت شود

- Parser نباید به Storage دسترسی مستقیم داشته باشد
- Storage نباید از وجود دستورها خبر داشته باشد
- Engine نباید فرمت ورودی کاربر را بداند

✅ ۲. داشتن ماژول‌های جداگانه برای هر کلاس

کدهای چسبیده به هم یا در یک فایل

✅ ۳. نام‌گذاری صحیح

- کلاس‌ها: PascalCase
- توابع و متغیرها: camelCase
- نام‌ها باید معنی‌دار باشند:
  - بد ❌ : `x`, `y`, `arr2`
  - خوب ✔️ : `studentList`, `transactionStack`

✅ ۴. تابع‌های کوتاه و مینیمال

- تابع نباید طولانی و چندکاره باشد
- حداکثر یک وظیفه مشخص برای هر تابع

✅ ۵. عدم تکرار کد (DRY)

- کپی‌پیست در چند کلاس ← کسر نمره

✅ ۶. استفاده از Interface و Abstraction

- مجموعه‌ها باید از طریق Interface مشترک مدیریت شوند
- نباید نوع کالکشن در لایه بالاتر مشخص باشد

✅ ۷. نبودن کدهای کامنت‌شده اضافی

- فقط کامنت‌های مفهومی و حرفه‌ای مجاز هستند.

⚠️ **توجه: این بخش برای اطمینان از تبدیل پروژه شما به یک کد قابل ارائه در رزومه یا گیت‌هاب است.**

🧮 بارم‌بندی پروژه (۲۰۰۰ نمره اصلی + ۶۰۰ امتیازی)

💾 ۱. Storage Layer — ۸۰۰

- ۳۰۰ نمره: ArrayCollection
- ۳۰۰ نمره: LinkedListCollection
- ۲۰۰ نمره: توابع جستجو و Aggregation

⚙️ ۲. Execution Engine — ۵۰۰

- ۳۰۰ نمره: تراکنش‌ها
- ۲۰۰ نمره: صف Batch

🔍 ۳. Parser — ۳۰۰

- ۱۵۰ نمره: پارس دستورات
- ۱۵۰ نمره: Import CSV

📊 ۴. گزارش فنی — ۴۰۰

- ۱۵۰ نمره: جدول Big O
- ۱۵۰ نمره: نمودارها
- ۱۰۰ نمره: سؤال‌های مفهومی

🏆 ۵. ۶۰۰ — امتیازی

- ۲۰۰ نمره: دیتابیس تحت شبکه
- ۱۵۰ نمره: ذخیره‌سازی دائمی
- ۲۵۰ نمره: فایل‌سیستم پیشرفته

🧼 ۶.(-۲۰۰) — Clean Code

- معماری لایه‌ای نادرست
- نام‌گذاری بد، توابع بزرگ، کد تکراری
- عدم استفاده از Interface/Abstraction مناسب
- ساختاردهی غلط پروژه یا تحویل فایل یک‌تکه
- کد اسپاگتی و عدم رعایت Clean Architecture