Documento de Descrição de Domínio

Packege para converter [JSON,CSV] para .DBF

Versão do Projeto: 1.0.0
Versão do Documento: 1.0

Empresa Cliente: ACDG
Empresa Responsável pelo Documento: ENVOLVE

Especialista de Domínio: GABRIEL VIEIRA SORIANO ADERALDO
Quality Assurance Lead: JORGE VICTOR
Product Manager: GABRIEL VIEIRA SORIANO ADERALDO
Tech Lead: BRENO COLAÇO
Product Owner (P.O): GABRIEL VIEIRA SORIANO ADERALDO

Data: March 28, 2025

Contents

1 Contexto do Sistema

Com o avanço das tecnologias de banco de dados e a ampla adoção de soluções baseadas em SQL e NoSQL, espera-se uma transição natural para formatos mais flexíveis e escaláveis. No entanto, muitos sistemas governamentais e corporativos ainda utilizam arquivos .DBF (dBase File) como meio principal para armazenamento e transferência de dados. Essa permanência se deve à forte dependência de sistemas legados, à complexidade da migração e à necessidade de interoperabilidade com aplicações antigas. Embora o formato .DBF seja uma tecnologia consolidada, sua utilização moderna apresenta desafios, principalmente na conversão e manipulação eficiente desses arquivos. A ausência de ferramentas atualizadas que permitam uma transição fluida para estruturas de dados contemporâneas impacta diretamente a eficiência dos sistemas que ainda dependem desse formato.

1.0.1 O que é um arquivo .DBF?

Criado na década de 1980 pelo software dBase, o .DBF foi amplamente adotado em aplicações que necessitavam de armazenamento tabular simples e acessível. Seu design permitiu que diversos sistemas gerenciassem registros estruturados de forma local, sem a necessidade de servidores de banco de dados. A compatibilidade do .DBF com tecnologias como FoxPro, Clipper, Visual dBase e xBase reforçou sua adoção em ambientes corporativos e governamentais. Sua estrutura de tabelas organizadas em colunas, associada à facilidade de acesso direto aos registros, tornou-o uma opção atrativa para sistemas de médio e grande porte durante o final do século XX.

1.0.2 Características do Formato .DBF

- **Cabeçalho**: Define os campos da tabela (nome, tipo de dado, tamanho).
- **Registros**: Contêm os dados armazenados, com tamanho fixo por linha.
- **Marcação de exclusão**: Permite que registros sejam "deletados" sem removê-los fisicamente, o que acelera operações.

Essa arquitetura, embora eficiente para aplicações standalone, apresenta limitações significativas em operações de manipulação em larga escala, dificultando a integração com sistemas modernos que dependem de bancos de dados relacionais ou não relacionais.

1.0.3 Exemplos de Sistemas Governamentais que Utilizam .DBF

Apesar das limitações, o .DBF continua sendo um componente essencial em diversas infraestruturas governamentais brasileiras, como:

- **SINAN (Sistema de Informação de Agravos de Notificação)**: Utilizado pelo Ministério da Saúde do Brasil para armazenar informações sobre doenças e agravos de notificação compulsória.
- **SIA/SUS (Sistema de Informações Ambulatoriais do SUS)**: Responsável pelo processamento de atendimentos ambulatoriais em unidades de saúde pública.
- **SEFIP (Sistema Empresa de Recolhimento do FGTS e Informações à Previdência Social)** Utilizado para envio de dados de FGTS e Previdência Social.
- **SPED (Sistema Público de Escrituração Digital)**: Algumas aplicações dentro do SPED Fiscal ainda permitem exportações em .DBF.
- **Sistemas Cartorários e Notariais**: Muitos cartórios brasileiros utilizam .DBF para armazenar registros de imóveis, certidões e transações. Sistemas de Prefeituras e Estados – Diversos sistemas de arrecadação tributária, controle de IPTU, alvarás e cadastros imobiliários operam com .DBF.

A permanência desses sistemas reflete a necessidade de ferramentas que facilitem a interoperabilidade entre .DBF e outras estruturas de dados mais flexíveis, como JSON, CSV e bancos relacionais.

1.0.4 Por Que o Governo Brasileiro Ainda Utiliza .DBF?

A continuidade do uso do .DBF no Brasil não é apenas uma questão de preferência tecnológica, mas sim uma consequência de fatores estruturais, incluindo:

- **Histórico de uso e legado**:

  - Muitas aplicações desenvolvidas entre os anos 1980 e 2000 foram construídas utilizando dBase, Clipper, FoxPro, entre outras tecnologias baseadas em .DBF.

  - Esses sistemas continuam sendo utilizados, pois a migração para bancos modernos é custosa e complexa.

- **Facilidade de Integração**:

  - O formato .DBF é simples de manipular e pode ser lido por diversas linguagens de programação sem necessidade de drivers complexos.

- O formato .DBF é simples de manipular e pode ser lido por diversas linguagens de programação sem necessidade de drivers complexos.

- **Baixo Custo e Independência de Banco de Dados**:

  - .DBF pode ser acessado sem um banco de dados SQL ou servidor dedicado, o que reduz os custos operacionais.

  - O governo brasileiro utiliza o formato .DBF em diversos sistemas para facilitar o armazenamento e a transmissão de dados entre diferentes órgãos e softwares.

Assim querendo resolver o problema de: Esses sistemas muitas vezes não possuem ferramentas modernas para manipulação de .DBF, tornando difícil a conversão para formatos mais flexíveis, como JSON e CSV. A ENVOLVE está propondo a a criação de uma biblioteca moderna e performática para conversão e manipulação de arquivos .DBF, utilizando Bun para maximizar a velocidade e eficiência na manipulação de arquivos binários. Diferente das soluções disponíveis, este sistema será modular e escalável, seguindo Domain-Driven Design (DDD) para garantir baixo acoplamento e facilidade de manutenção.

1.0.5 Motivação para o Uso de JavaScript/Bun

Embora linguagens como Go e Rust ofereçam alto desempenho, optamos por JavaScript/Bun devido a fatores críticos como ecossistema, suporte à web e acessibilidade para desenvolvedores. Além disso, consideramos Deno, mas optamos por Bun por razões específicas detalhadas a seguir.

- **Falta de Bibliotecas Atualizadas para .DBF em Node.js/Deno**:

  - **Node.js**: As bibliotecas existentes para manipulação de .DBF estão desatualizadas ou limitadas.

  - **Deno**: Embora existam bibliotecas como deno-dbf-file, elas ainda estão em estágios iniciais de desenvolvimento e podem não oferecer todas as funcionalidades necessárias.

- **Popularidade do JavaScript em APIs e Soluções Web**:

  - JavaScript é amplamente utilizado no desenvolvimento web, tornando a biblioteca mais acessível para a comunidade de desenvolvedores.

  - A ENVOLVE já possui experiência com JavaScript e TypeScript, o que acelera o desenvolvimento e a manutenção do sistema.

- **Ecossistema de Módulos e Ferramentas**:

- possui um ecossistema robusto com inúmeras bibliotecas e frameworks, facilitando a integração e expansão da funcionalidade.

- Ferramentas como Bun permitem a criação de aplicações performáticas e escaláveis, garantindo alta eficiência no processamento de arquivos .DBF.

- Bun oferece desempenho superior ao Node.js e comparável ao Deno, sendo adequado para operações de E/S intensivas.

- A biblioteca será otimizada para manipulação de arquivos .DBF, garantindo alta velocidade e eficiência na conversão e leitura de dados.

- **Manutenção e Facilidade de Distribuição**:

  - permite distribuição simplificada através do NPM, facilitando a instalação e atualização da biblioteca.

1.0.6 Justificativa para a Escolha de Bun

- Há uma lacuna no ecossistema JavaScript para bibliotecas modernas de manipulação de .DBF.
- JavaScript é amplamente utilizado, tornando a biblioteca mais acessível e incentivando a adoção.
- JavaScript possui um ecossistema rico que facilita a integração e expansão da funcionalidade.
- Bun oferece desempenho suficiente para as necessidades do projeto, equilibrando velocidade e facilidade de desenvolvimento.
- A distribuição via NPM/Bun simplifica a instalação e atualização da biblioteca.
- Bun é mais rápido que Node.js e oferece desempenho comparável ao Deno.
- Essencial para manipulação eficiente de arquivos .DBF.
- oferece uma experiência de desenvolvimento simplificada, acelerando o processo de criação e manutenção da biblioteca.
- Desenvolvedores familiarizados com JavaScript podem adotar Bun rapidamente, reduzindo o tempo de adaptação.

1.1 Arquitetura do Sistema

Visão Geral da Arquitetura: A biblioteca seguirá uma arquitetura modular, baseada em Domain-Driven Design (DDD) e organizada em três camadas principais:

- **Camada de Domínio (CORE DOMAIN) "core/"**: Contém as regras de negócio, garantindo a estrutura correta do .DBF.
- **Camada de Aplicação ((Application Layer) "application/")**: Orquestra os casos de uso e a conversão de formatos.
- **Camada de Infraestrutura (Infrastructure Layer): "infrastructure/"**: Responsável por persistência, manipulação de arquivos e parsers, mantendo-se desacoplada para permitir substituições futuras.

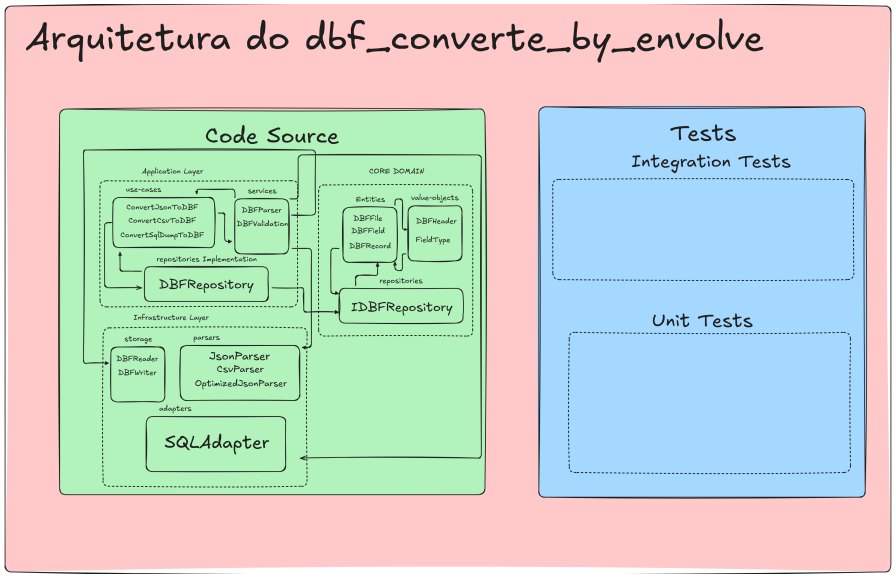

Figure 1: Arquitetura do Sistema

1.2 CORE DOMAIN - DBFStructure

O domínio ‘DBFStructure‘ é responsável por definir, validar e gerenciar arquivos no formato ‘.DBF‘, um dos formatos de banco de dados mais utilizados historicamente. Apesar de ser um padrão antigo, ele ainda é amplamente adotado em sistemas governamentais, softwares contábeis e aplicações legadas devido à sua simplicidade, compatibilidade e estrutura binária eficiente. No entanto, essa mesma rigidez estrutural impõe desafios na manipulação desses arquivos, exigindo um

modelo bem definido para garantir consistência, compatibilidade e confiabilidade ao armazenar e recuperar informações.

O ‘DBFStructure‘ foi projetado para oferecer uma solução modular e extensível para manipulação de arquivos ‘.DBF‘, permitindo que diferentes aplicações trabalhem com esses dados sem precisar lidar diretamente com a complexidade do formato. Esse domínio é responsável por definir a estrutura inicial dos arquivos, gerenciar a manipulação de registros e validar a integridade estrutural, garantindo que todas as operações respeitem as regras do ‘.DBF‘ e evitem corrupção dos dados.

Para garantir a separação de responsabilidades, o ‘DBFStructure‘ foi organizado em **Bounded Contexts**, cada um com um papel específico na manipulação dos arquivos ‘.DBF‘. Esses contextos garantem que operações distintas, como definição da estrutura, manipulação de registros e validação, sejam tratadas de forma isolada, minimizando dependências e permitindo evolução independente de cada módulo.

Os principais ‘Bounded Contexts‘ do domínio ‘DBFStructure‘ são:

- **DBFFileDefinition**: Responsável pela criação e definição da estrutura do ‘.DBF‘, incluindo seu cabeçalho (‘DBFHeader‘), campos (‘DBFField‘) e regras de estruturação. Esse contexto também gerencia a adição, alteração e remoção de campos dentro do arquivo, garantindo que qualquer modificação estrutural seja validada antes de ser aplicada. **Qualquer modificação na estrutura (‘DBFField‘) resultará na exclusão automática de todos os registros (‘DBFRecord‘) existentes no arquivo.**

- **DBFRecordManagement**: Responsável pelo gerenciamento dos registros (‘DBFRecord‘) dentro de um arquivo ‘.DBF‘, incluindo operações de inserção, remoção e edição de registros. **Esse contexto depende da estrutura definida pelo ‘DBFFileDefinition‘ e só pode operar sobre arquivos ‘.DBF‘ previamente validados.**

- **DBFValidation**: Responsável pela validação estrutural e de dados dos arquivos ‘.DBF‘, garantindo que qualquer modificação realizada dentro do ‘DBFFileDefinition‘ ou ‘DBFRecordManagement‘ esteja dentro das regras do formato ‘.DBF‘, assegurando consistência e integridade.

- **DBFExport**: Responsável pela conversão da estrutura interna do ‘.DBF‘ para um arquivo binário ‘.DBF‘ válido, garantindo que os dados e a estrutura sejam corretamente escritos em disco, permitindo que o arquivo seja utilizado por sistemas externos.

A modularidade do ‘DBFStructure‘ permite que cada um desses contextos evolua independentemente, possibilitando melhorias contínuas sem impactar funcionalidades já existentes. Além disso, essa abordagem facilita a integração com infraestruturas externas, como conversores de formatos (‘JSON‘, ‘CSV‘, ‘SQL Dump‘), sistemas de banco de dados relacionais e APIs que precisem manipular ‘.DBF‘.

O diferencial do ‘DBFStructure‘ não está apenas em fornecer um modelo robusto para lidar com ‘.DBF‘, mas também em permitir sua evolução gradual. O design modular do domínio possibilita suporte a novos formatos, versões mais recentes do ‘.DBF‘, e melhorias na validação, tornando os arquivos ‘.DBF‘ mais acessíveis para integração com sistemas modernos sem perda de confiabilidade ou compatibilidade.

1.2.1 Objetivos

- Definir a Estrutura de um Arquivo ‘.DBF‘, garantindo que cabeçalhos, campos e registros sigam as regras do formato.
- Gerenciar Registros (‘DBFRecord‘) do ‘.DBF‘, permitindo inserção, edição e remoção sem comprometer a estrutura.
- Modificar a estrutura do ‘.DBF‘ (‘DBFField‘), permitindo a adição e remoção de colunas. **Qualquer alteração na estrutura resulta na exclusão automática de todos os registros (‘DBFRecord‘).**
- Validar a Estrutura do ‘.DBF‘ para garantir integridade e compatibilidade com sistemas que utilizam esse formato.
- Assegurar Consistência e Compatibilidade do Arquivo, respeitando os limites de tamanho, alinhamento e regras de versionamento.
- Facilitar a Integração com Outras Tecnologias, permitindo exportação e conversão para formatos modernos.

1.2.2 Futuros Objetivos

- Suporte a Novos Formatos de Exportação (‘XML‘, ‘Parquet‘, ‘YAML‘).
- Suporte a Campos do Tipo Memo (‘M‘), utilizados em versões mais recentes do ‘.DBF‘ para armazenamento de textos longos.
- Suporte a Diferentes Versões do ‘.DBF‘ (dBase IV (0x04), dBase V (0x05), FoxPro (0x30)).

- Melhorias na Validação e Correção Automática de Estruturas ‘.DBF‘ corrompidas.
- Integração com Banco de Dados Relacionais, permitindo conversões diretas entre ‘.DBF‘ e ‘PostgreSQL‘, ‘MySQL‘, ‘SQLite‘ e outros.

1.3 Bounded Context

1. DBFFileDefinition: Define a estrutura inicial do arquivo .DBF, incluindo o cabeçalho (DBFHeader) e os campos (DBFField). Este contexto é responsável pela criação da estrutura base do arquivo e não gerencia a adição, remoção ou alteração de campos.

2. DBFFieldManagement: Gerencia a adição, alteração e remoção de campos (DBFField) dentro do arquivo .DBF. Assegura que qualquer modificação estrutural seja validada e aplicada corretamente. Qualquer modificação na estrutura de campos resulta na remoção lógica dos registros (DBFRecord).

3. DBFRecordManagement: Gerencia a manipulação dos registros (DBFRecord) dentro do arquivo .DBF, permitindo a inserção, edição, remoção lógica e recuperação de registros. Opera sobre arquivos .DBF com estrutura previamente validada.

4. DBFValidation: Valida a estrutura e os registros do .DBF. Garante que qualquer modificação nos contextos DBFFileDefinition, DBFFieldManagement e DBFRecordManagement esteja em conformidade com as regras do formato .DBF.

5. DBFExport: Converte a estrutura interna do .DBF para um arquivo binário .DBF válido e o armazena, garantindo a correta escrita dos dados e da estrutura para uso externo.

6. DBFImport: Importa dados de fontes externas (e.g., CSV, JSON) para a estrutura interna do .DBF. Garante que os dados importados estejam em conformidade com a estrutura definida e cria os registros (DBFRecord) correspondentes.

1.4 Definição do BOUNDED CONTEXT: DBFFileDefinition

O DBFFileDefinition é o núcleo responsável por estabelecer a estrutura inicial de um arquivo .DBF. Ele assegura que a estrutura base seja válida e compatível com as regras do formato .DBF. Esse contexto gerencia a criação da "casca" do arquivo, preparando-o para operações futuras, como inserção de registros (DBFRecord) ou exportação (DBFExport). Os arquivos .DBF seguem um formato binário rígido, onde cada componente:
- DBFHeader (cabeçalho)
- DBFField (campos)
- DBFRecord (registros)

precisa estar corretamente alinhado e estruturado para garantir compatibilidade com sistemas que processam esse tipo de dado. Um erro na definição estrutural pode tornar o arquivo ilegível para outros sistemas, tornando a configuração correta da estrutura um aspecto crítico. Esse contexto atua diretamente na criação da estrutura inicial de um .DBF. Ele define regras para nomes de campos, tipos de dados, tamanho dos registros e alinhamento da memória, garantindo que a estrutura seja construída corretamente desde o início. É importante ressaltar que, nesse estágio, as listas de campos (List<DBFField>) e registros (List<DBFRecord>) são inicializadas vazias.

Important!

- O DBFFileDefinition é responsável pela criação da "casca" inicial do arquivo .DBF.
- Neste contexto, as listas de campos (List<DBFField>) e registros (List<DBFRecord>) são inicializadas vazias.
- A validação completa do arquivo .DBF só deve ocorrer após a adição de dados e é responsabilidade de outro contexto (DBFValidation).

O DBFFileDefinition será modelado como um agregado (Aggregate Root), onde o DBFFile atuará como a raiz do agregado. Esse agregado encapsula toda a estrutura inicial do .DBF. O DBFFile conterá as seguintes estruturas:

- DBFHeader: Metadados do .DBF, incluindo versão, data de modificação, contagem de registros e deslocamento do primeiro registro.
- List<DBFField>: Lista de campos (DBFField) que definem a estrutura do .DBF, incluindo nome, tipo de dado e tamanho.
- List<DBFRecord>: Lista de registros (DBFRecord), que é inicializada vazia.

Esse agregado garantirá que:

- A estrutura inicial do .DBF seja consistente e válida.
- O DBFFileDefinition não gerencia a adição, remoção ou alteração de campos, nem a manipulação de registros. Essas são responsabilidades de outros Bounded Contexts.

1.5 Organização do DBFFileDefinition

O DBFFileDefinition é responsável pela criação da estrutura inicial do .DBF, garantindo que ele seja construído de maneira válida e coerente. Ele tem como foco exclusivo a definição da "casca" do arquivo, ou seja, a estrutura base, preparando-o para receber dados e ser utilizado por outros contextos. Este contexto não gerencia a modificação de campos (DBFField) ou a manipulação de registros (DBFRecord).

Responsabilidades Exclusivas:

1. O DBFFileDefinition não gerencia a adição, remoção ou alteração de campos (DBFField).
2. O DBFFileDefinition não gerencia a manipulação de registros (DBFRecord).
3. Essas responsabilidades são delegadas aos Bounded Contexts DBFFieldManagement e DBFRecordManagement, respectivamente.

1.6 Como Esse Contexto Será Implementado?

O DBFFileDefinition será modelado como um agregado (Aggregate Root), onde o DBFFile atuará como a raiz do agregado. Esse agregado encapsula a estrutura inicial do .DBF, representando a "casca" do arquivo.

1.6.1 Tarefas Principais do DBFFileDefinition

O DBFFileDefinition tem como tarefas principais a criação da estrutura inicial do .DBF e a preparação para sua utilização por outros contextos.

Gerenciamento da Estrutura Inicial:

- Criar um novo arquivo .DBF, definindo o cabeçalho (DBFHeader) e inicializando a lista de campos (List<DBFField>) e a lista de registros (List<DBFRecord>) como vazias.
- Garantir que o tamanho do cabeçalho seja calculado corretamente.
- Determinar a posição (offset) do primeiro registro dentro do arquivo.

Preparação para Outros Contextos:

- Fornecer a "casca" do .DBF pronta para ser populada com dados e validada (DBFValidation).

- Garantir que a estrutura inicial definida seja compatível com a exportação (DBF-Export).

1.6.2 Tipos de dados do DBFFileDefinition

Esta seção detalha os tipos de dados utilizados no contexto DBFFileDefinition, suas propriedades e as regras de negócio associadas

1.6.3 Informações gerais do DBFFile

- **Nome do dado**: ***DBFFile***
- **Tipo do dado**: ***Aggregate Root***
- **Descrição do dado**: O ***DBFFile*** é a raiz do agregado e centraliza todas as regras de criação e manipulação da estrutura do ***.dbf***. Ele encapsula o cabeçalho: ***DBFHeader***, a lista de campos:***List<DBFField>*** e os registros:***List<DBFRecord>***, garantindo que nenhuma modificação ocorra de maneira inconsistente.

Atributos do Aggregate Root DBFFile

Atributo	Tipo	Descrição
name	string	Nome do arquivo .DBF.
header	DBFHeader	Metadados do .DBF (versão, data, quantidade de registros).
fields	List<DBFField>	Lista de colunas (campos da tabela).
records	List<DBFRecord>	Lista de registros armazenados dentro do .DBF.
recordSize	number	Define o tamanho dos Registros deste dbf, sendo calculado pela soma em bytes dos CAMPOS do arquivo

Table 1: Atributos do DBFFile e suas descrições

1.6.4 Metodos que estão dentro do DBFFile

- **create**: Método responsável por criar a instancia do DBFFile, A implementação não amarra como será feita, caso queira usar um factory ou um builder, a seu criterio. Sempre uma nova instancia for criada, lembre-se que os campos fields e records devem ser listas vazias. O DBFHeader não pode está vazia.

- **calculateRecordSize**: Método responsável por calcular o tamanho da soma de todos os campos: ***DBFField*** em bytes.
- **validateHeaderSize**: Método responsável por validar o tamanho do cabeçalho, o tamanho do cabeçalho deve ser múltiplo de 32 bytes. Ou seja, caso o tamanho do cabeçalho em bytes for um numero que não é multiplo de 32, o sistema tem que adicionar a quantidade de bytes com zeros até que o tamanho seja múltiplo de 32.
- **addDBFField**: Método responsável por adicionar um novo campo ***DBFField*** à lista de campos do DBFFile. O campo deve ser validado antes de ser adicionado.
- **addDBFRecord**: Método responsável por adicionar um novo registro ***DBFRecord*** à lista de registros do DBFFile. O registro deve ser validado antes de ser adicionado.

1.6.5 Informações Gerais do DBFField

Nome do dado: ***DBFField***
Tipo do dado: Entidade
Descrição do dado: Cada DBFField representa um campo dentro do ***.bdf***, definindo nome, tipo, tamanho e alinhamento dos dados armazenados.

Atributos da entidade do DBFField

Atributo	Tipo	Descrição
name	string	Nome do campo da tabela
type	enum	Um enumerador com os valores permitidos: ver os valores na tabela do enum abaixo.
size	number	Tamanho do campo em bytes.
decimal	number	Aqui fica qual o número de casas decimais que será colocado, quando os registros forem escritos. Exemplo: Caso você queira o numero 12 e coloque no tamanho do campo sendo 2, está tudo bem, pois o binario será: 0x31 0x32. Porém se for colocado no tamanho do campo 5 bytes e só dois decimais, ai a representação do numero será: 0x31 0x32 0x2E 0x30 0x30, Pois obrigatoriamente quando é colocado só 2 decimais o resto será flutuante, então será 1 caracter para o ponto e outros dois para zeros

Table 2: Atributos do DBFField e suas descrições

Enumerator do campo type do DBFField

Atributo	Tipo	Descrição
C	Character	É um tipo que aceita textos, porém com o tamanho máximo sendo de 10 bytes
N	Numeric	É um tipo que aceita numeros em geral, tanto decimais, quanto inteiros, porém com um tamanho máximo de 19 bytes
F	Float	É um tipo que aceita as mesmas regras do Numeric, porém é para numeros flutuantes com menor precisão
L	Logical	Booleano (T ou F, para verdadeiro ou falso). Sendo seu maximo 1 byte
D	Date	Data no formato YYYYMMDD (8 caracteres).

Table 3: Todos os valores possiveis que podem ser escolhidos para o campo type do ***DBFField***

1.6.6 Metodos que estão dentro do DBFFile

- **create**: Método responsável por criar a instancia do ***DBFField***, Ou seja ele cria um BDFFiel válido para ser colocado dentro da lista de ***DBFField***.
- **validateNameField**: Método responsável por validar o nome do campo, o nome do campo não pode ser vazio, e nem pode ter mais de 10 caracteres.
- **calculateSize**: Calcula o tamanho do campo em bytes, baseado no tipo de dado. Caso não contenha já registros, se sim pegue o maior registro deste campo e coloque como o tamanho.
- **validateDecimal**: Calcula se o valor de decimais é um numero inteiro válido, seguindo a formula:

  $I = P - (D + 1)$

  Onde:

  - I é o número máximo de dígitos inteiros.
  - P é o tamanho total do campo (quantidade máxima de caracteres permitidos).
  - D é o número de casas decimais.

  Exemplos práticos:

  - Suponha que $P = 6$ e $D = 2$.
  - Aplicando a fórmula:
    $$I = 6 - (2 + 1) = 3$$
  - Isso significa que o número pode ter, no máximo, **3 dígitos inteiros**.

  Explicação detalhada:

  - O valor de $P = 6$ indica que o campo pode armazenar **até 6 bytes** no total.
  - Além disso, o próprio **ponto decimal (‘.‘) ocupa 1 byte**, representado em binário como ***0x2E***.
  - Isso deixa **3 bytes disponíveis** para os dígitos inteiros.
  - Como cada número inteiro ocupa **1 byte por caractere**, o maior número que pode ser armazenado é **999**.
  - O número **1000** não cabe, pois precisaria de **4 bytes para os dígitos inteiros**, excedendo o limite permitido.

$$I \leq P - (D + 1), \quad seD > 0 \tag{1}$$

$$I \leq P, \quad seD = 0 \tag{2}$$

Figure 2: Fórmula para cálculo do número máximo de dígitos inteiros

1.6.7 Informações Gerais do DBFRecord

Nome do dado: ***DBFRecord***
Tipo do dado: Entidade
Descrição do dado: Cada ***DBFRecord*** representa uma linha de dados dentro do ***.bdf***, contendo valores para cada ***DBFField*** definido. Os registros devem seguir estritamente o formato dos campos ***DBFField*** definidos na estrutura do ***.bdf***

Atributos da entidade do DBFRecord

Atributo	Tipo	Descrição
value	Map<string,any>	Estrutura de chave/valor que armazena os dados de um registro, onde a chave é o nome do campo (DBFField.name) e o valor é o dado armazenado.
isDeleted	boolean	Indica se o registro foi logicamente marcado como excluído (0x2A).

Table 4: Atributos do DBFRecord e suas descrições

1.6.8 Metodos que estão dentro do DBFRecord

- **create**: Metodo que cria um DBFRecord, ou seja ele cria uma instancia de ***DBFRecord*** deixando pronto para ser inserido na lista de ***DBFRecord*** dentro do ***DBFFile***.
- **validateValue**: Método responsável por validar o valor do registro, verificando se ele está de acordo com o tipo e tamanho definido no ***DBFField*** correspondente.

1.6.9 Informações Gerais do DBFHeader

Nome do dado: ***DBFHeader***
Tipo do dado: Entidade

Descrição do dado: Cada ***DBFHeader*** representa o cabeçalho de um arquivo ***.dbf***, contendo informações gerais sobre o arquivo como versão, data da última modificação, número total de registros e a posição do primeiro registro. Esses dados são fundamentais para interpretação correta do conteúdo do ***.dbf***.

Atributos da entidade do DBFHeader

Atributo	Tipo	Descrição
version	enum	Versão do arquivo '.dbf'. Geralmente '0x03', que representa a versão dBase III.
lastUpdated	Date	Data da última modificação do arquivo.
recordCount	number	Quantidade total de registros armazenados no arquivo ('DBFRecord').
firstRecordOffset	number	Offset (posição em bytes) do primeiro registro no arquivo. Deve ser múltiplo de 32.

Table 5: Atributos do DBFHeader e suas descrições

Versões conhecidas do formato .DBF

	Identificador	Descrição
0x02	FoxBASE	Versão antiga usada pelo FoxBASE. Compatível com dBase II.
0x03	dBase III	Versão padrão mais comum, usada em muitos arquivos '.dbf'. Suporte a dados sem memo.
0x30	Visual FoxPro	Arquivo Visual FoxPro sem campos 'memo'. Suporta novos tipos de dados.
0x31	Visual FoxPro + Memo	Igual ao 0x30, mas com suporte a campos 'memo'.
0x83	dBase III + Memo	Variante da versão dBase III que inclui suporte a campos 'memo'.
0x8B	dBase IV + Memo	Versão do dBase IV com campos 'memo'.
0xF5	FoxPro + Memo	Versão do FoxPro com suporte a campos 'memo'.
0xFB	FoxBASE + Memo	Versão do FoxBASE com campos 'memo'.

Table 6: Códigos de versão do '.dbf' e suas descrições

1.6.10 Regras de negócio do DBFHeader

- **Validação de offset**: O valor de *firstRecordOffset* deve ser sempre múltiplo de 32 bytes.
- **Compatibilidade de versão**: O valor de *version* deve ser compatível com a versão esperada (‘0x03‘ para dBase III).
- **Atualização de registros**: O campo *recordCount* deve ser atualizado corretamente sempre que um *DBFRecord* for adicionado ou removido.

1.7 Diagrama de Classes do DBFFileDefinition

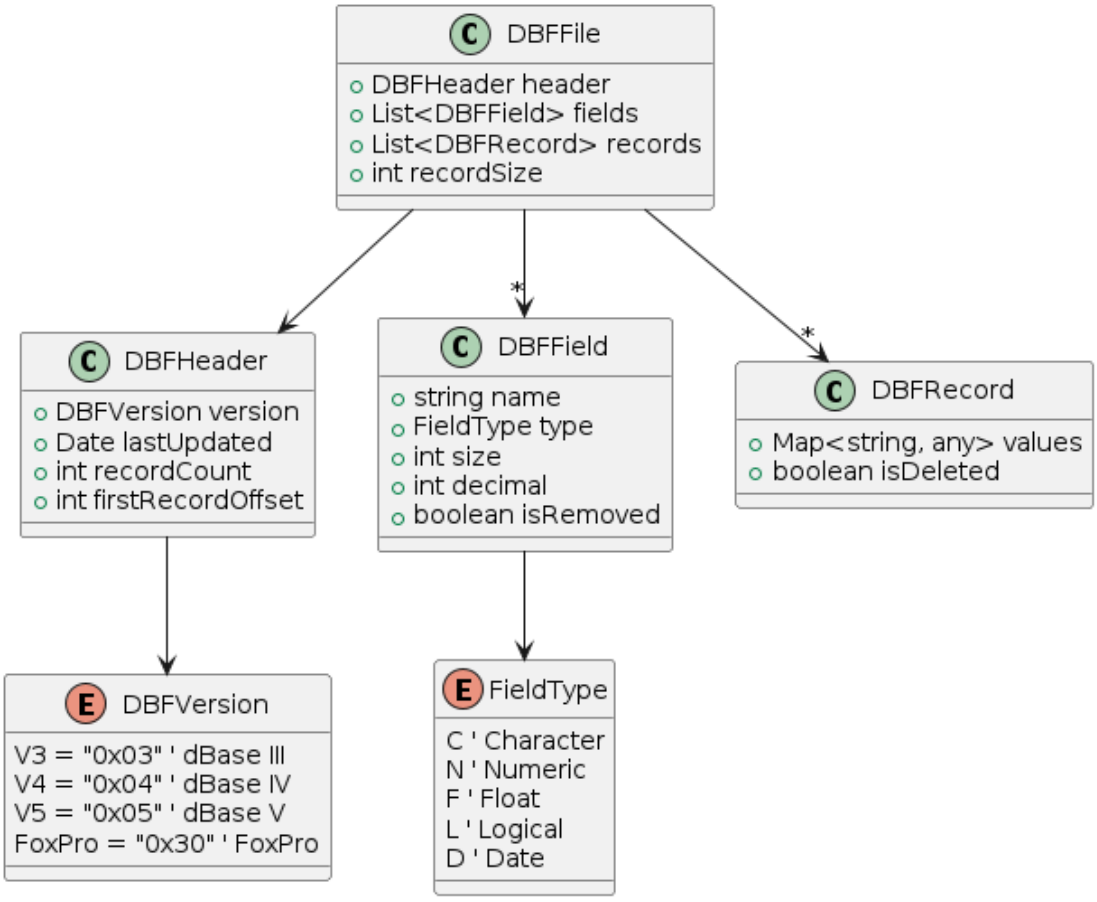

Figure 3: Diagrama de Classes do Bounded Context: DBFFileDefinition

1.8 BDD do caso de uso: CreateDBFFileBaseV3

1.8.1 Feature: Criar um objeto DBFFile válido

Como um usuário ou sistema
Quero criar um objeto **DBFFile** contendo:
- **DBFHeader**
- **List<DBFField>**
- **List<DBFRecord>**
inicializados corretamente Para que ele possa ser utilizado posteriormente por outros casos de uso

1.8.2 Background

Given que **DBFFile** representa um arquivo '.DBF' na memória
And DBFFile deve ser criado apenas com:
- **DBFHeader**
- **List<DBFField>**
- **List<DBFRecord>**
todos inicializados corretamente

1.8.3 Scenario: Criar um objeto DBFFile com campos e registros válidos

Given que o sistema inicia a criação de um **DBFFile**
When o sistema cria um **DBFHeader**
And o **DBFHeader** contém a versão padrão do '.DBF' ('0x03')
And o **DBFHeader** contém a data atual como última modificação
And o **DBFHeader** tem o número total de registros inicializado como zero
And o **DBFHeader** calcula a posição inicial onde os registros serão armazenados
And o sistema inicializa uma **List<DBFField>** vazia
And o sistema inicializa uma **List<DBFRecord>** vazia
Then o sistema associa o **DBFHeader**, **List<DBFField>** e **List<DBFRecord>** ao **DBFFile**
And o sistema retorna um **DBFFile** pronto para ser manipulado

1.8.4 Scenario: Criar um DBFFile sem registros iniciais

Given que o sistema inicia a criação de um **DBFFile**
When o sistema cria um **DBFHeader**

And o sistema inicializa uma **List<DBFField>** vazia
And a **List<DBFRecord>** está vazia, pois não há registros no momento da criação
Then o sistema retorna um **DBFFile** válido contendo apenas cabeçalho e campos

1.8.5 Scenario: Falha ao criar o DBFHeader

Given que ocorreu um erro ao criar o **DBFHeader**
When o sistema tenta associar o **DBFHeader** ao **DBFFile**
Then o sistema deve lançar um erro **"Erro: Falha ao criar o cabeçalho do DBF"**
And o sistema não deve criar um **DBFFile**

1.8.6 Scenario: Falha ao inicializar a lista de campos (‘DBFField‘)

Given que ocorreu um erro ao criar a **List<DBFField>** vazia
When o sistema tenta associar a lista de campos ao **DBFFile**
Then o sistema deve lançar um erro **"Erro: A lista de campos deve ser inicializada vazia"**
And o sistema não deve criar um **DBFFile**

1.8.7 Scenario: Falha ao inicializar a lista de registros (‘DBFRecord‘)

Given que ocorreu um erro ao criar a **List<DBFRecord>** vazia
When o sistema tenta associar a lista de registros ao **DBFFile**
Then o sistema deve lançar um erro **"Erro: A lista de registros deve ser inicializada vazia"**
And o sistema não deve criar um **DBFFile**

1.8.8 Scenario: Falha ao definir a versão do ‘.DBF‘

Given que o sistema inicia a criação de um **DBFFile**
When o sistema cria um **DBFHeader**
And a versão do ‘.DBF‘ não pode ser definida corretamente
Then o sistema deve lançar um erro **"Erro: Versão do ‘.DBF‘ inválida"**
And o sistema não deve criar um **DBFFile**

1.8.9 Scenario: Falha ao definir a data da última modificação

Given que o sistema inicia a criação de um **DBFFile**
When o sistema cria um **DBFHeader**
And ocorre um erro ao definir a data da última modificação
Then o sistema deve lançar um erro **"Erro: Data de modificação inválida"**
And o sistema não deve criar um **DBFFile**

1.8.10 Scenario: Falha ao calcular o offset do primeiro registro

Given que o sistema inicia a criação de um **DBFFile**
When o sistema cria um **DBFHeader**
And ocorre um erro ao calcular a posição inicial do primeiro registro
Then o sistema deve lançar um erro **"Erro: Falha ao calcular o offset do primeiro registro"**
And o sistema não deve criar um **DBFFile**

1.9 Definição do BOUNDED CONTEXT: DBFFieldManagement

O **DBFFieldManagement** é responsável por gerenciar a estrutura dos campos (**DBFField**) dentro de um arquivo **.DBF**, permitindo **adição, modificação e remoção** de campos.

Os campos (**DBFField**) definem a estrutura do **.DBF**, ou seja, quais informações podem ser armazenadas nos registros (**DBFRecord**). Como o formato **.DBF** possui alinhamento fixo, **qualquer modificação nos campos exige a remoção de todos os registros (DBFRecord) existentes**, pois os dados podem ficar desalinhados e ilegíveis para outros sistemas.

Esse contexto não gerencia registros (DBFRecord), pois essa responsabilidade pertence ao **DBFRecordManagement**.

1.10 Como Esse Contexto Será Implementado?

O **DBFFieldManagement** será implementado como um conjunto de operações dentro do **DBFFile** (Aggregate Root). Ele permitirá modificar a estrutura dos campos (**DBFField**), garantindo que as mudanças respeitem as regras do **.DBF** e que os registros (**DBFRecord**) sejam excluídos automaticamente quando necessário.

IMPORTANTE:

- Qualquer **adição, modificação ou remoção de um DBFField** exige a **recriação da estrutura do .DBF**, pois os registros (**DBFRecord**) dependem diretamente dos campos.
- Após qualquer alteração estrutural, **todos os registros (DBFRecord) serão apagados automaticamente** para evitar inconsistências.

1.11 Responsabilidades do DBFFieldManagement

- Adicionar um novo campo (**DBFField**) sem comprometer a estrutura do **.DBF**.
- Modificar um campo (**DBFField**) garantindo que a alteração respeite as regras do formato.
- Remover um campo (**DBFField**) e garantir que a estrutura permaneça válida.
- Assegurar que todas as modificações passem por validação (**DBFValidation**) antes de serem aplicadas.
- Atualizar metadados do **DBFHeader** (ex: **recordCount**) para refletir mudanças na estrutura.
- Garantir que qualquer modificação nos campos (**DBFField**) resulte na remoção automática dos registros (**DBFRecord**).

1.12 Tipos de Dados no DBFFieldManagement

O **DBFFieldManagement** lida diretamente com os seguintes componentes:

1.12.1 DBFFile (Aggregate Root)

Descrição: O **DBFFile** encapsula toda a estrutura do **.DBF**, incluindo os campos (**List<DBFField>**) e registros (**List<DBFRecord>**).

Atributo	Tipo	Descrição
header	DBFHeader	Contém metadados do **.DBF** (versão, data, contagem de registros).
fields	List<DBFField>	Contém todos os campos do **.DBF**.
records	List<DBFRecord>	Contém todos os registros armazenados.
recordSize	number	Define o tamanho fixo de cada registro.

Regras de Negócio:

1. O **.DBF** deve ter pelo menos um **DBFField** para ser válido.
2. Qualquer alteração em **DBFField** deve apagar todos os **DBFRecord** para evitar registros desalinhados.
3. O **recordSize** deve ser recalculado sempre que um campo for adicionado, alterado ou removido.

1.12.2 DBFField

Descrição: Cada **DBFField** representa uma coluna dentro do **.DBF**, definindo o nome, tipo, tamanho e alinhamento dos dados.

Atributo	Tipo	Descrição
name	string	Nome do campo (máx. 10 caracteres).
type	enum	Define o tipo de dado armazenado.
size	number	Tamanho do campo em bytes.
decimal	number	Número de casas decimais (apenas para **N**).
isRemoved	boolean	Indica se o campo foi marcado como "removido".

Regras de Negócio:

1. O nome do campo não pode exceder 10 caracteres.
2. O tipo do campo deve ser válido (**C, N, F, L, D**).
3. Se o campo for do tipo **Numérico (N)**, deve validar casas decimais (mín. 1, máx. 19 bytes).
4. Se um campo for removido (**isRemoved**), ele não pode mais ser usado em novos registros.

1.13 Relação do DBFFieldManagement com Outros Contextos

1.13.1 DBFFileDefinition

Tipo de Relação: Shared Kernel
Descrição: O **DBFFileDefinition** define a estrutura inicial do **.DBF**, enquanto o **DBFFieldManagement** permite modificá-la. Como ambos compartilham o mesmo **DBFFile**, mudanças devem ser feitas com **validação rigorosa**.

1.13.2 DBFRecordManagement

Tipo de Relação: Customer/Supplier
Descrição: Como os registros (**DBFRecord**) dependem dos campos (**DBFField**), qualquer modificação estrutural exige que todos os registros sejam apagados.

1.13.3 DBFValidation

Tipo de Relação: Open-Host Service
Descrição: Sempre que um campo (**DBFField**) for **adicionado, alterado ou removido**, o **DBFValidation** deve garantir que a estrutura do **.DBF** permaneça válida.

1.13.4 DBFExport

Tipo de Relação: Published Language
Descrição: O **DBFFieldManagement** define a estrutura do **.DBF**, garantindo que ele possa ser corretamente exportado pelo **DBFExport**.

1.14 Conclusão

O **DBFFieldManagement** desempenha um papel crucial dentro do **DBFStructure**, pois permite modificar a estrutura do **.DBF** sem comprometer sua integridade. Ele **não gerencia registros (DBFRecord)**, mas garante que qualquer mudança nos campos (**DBFField**) seja segura e validada.

1.14.1 Scenario: Adicionar um novo campo (‘DBFField‘) ao ‘.DBF‘

Given que o sistema possui um ‘DBFFile‘ válido com uma ‘List<DBFField>‘ existente
When o usuário solicita a adição de um novo ‘DBFField‘
Then o sistema deve validar se o nome do campo não excede 10 caracteres
And o sistema deve validar se o tipo (‘type‘) do campo pertence ao conjunto permitido ‘C, N, F, L, D‘
And o sistema deve validar que o nome do campo não está duplicado na ‘List<DBFField>‘
And o sistema adiciona o novo ‘DBFField‘ à ‘List<DBFField>‘
And o sistema remove todos os registros (‘DBFRecord‘) do ‘DBFFile‘ para evitar inconsistências
And o sistema atualiza o ‘DBFFile‘ e chama a validação (‘DBFValidation‘)
And o sistema retorna o ‘DBFFile‘ atualizado

Scenario: Falha ao adicionar um campo com nome inválido Given que o usuário tenta adicionar um ‘DBFField‘ com um nome maior que 10 caracteres
When o sistema valida o nome do campo
Then o sistema deve lançar um erro ‘"Erro: O nome do campo não pode exceder 10 caracteres"‘
And o sistema não deve modificar a ‘List<DBFField>‘

Scenario: Falha ao adicionar um campo com tipo inválido Given que o usuário tenta adicionar um ‘DBFField‘ com um tipo não suportado (‘M‘)
When o sistema valida o tipo do campo
Then o sistema deve lançar um erro ‘"Erro: O tipo ’M’ não é suportado pelo formato dBase III"‘
And o sistema não deve modificar a ‘List<DBFField>‘

Scenario: Falha ao adicionar um campo duplicado Given que já existe um ‘DBFField‘ chamado ‘"Nome"‘ na ‘List<DBFField>‘
When o usuário tenta adicionar um novo ‘DBFField‘ com o mesmo nome ‘"Nome"‘
Then o sistema deve lançar um erro ‘"Erro: O nome do campo já existe na estrutura do .DBF"‘
And o sistema não deve modificar a ‘List<DBFField>‘

1.14.2 Scenario: Modificar um campo (‘DBFField‘) existente

Given que o ‘DBFFile‘ contém um campo chamado ‘"Idade"‘ na ‘List<DBFField>‘
When o usuário solicita a modificação do campo ‘"Idade"‘
Then o sistema deve validar que o campo existe na ‘List<DBFField>‘
And o sistema deve validar se o novo nome não excede 10 caracteres
And o sistema deve validar se o novo tipo (‘type‘) pertence ao conjunto permitido ‘C, N, F, L, D‘
And o sistema deve atualizar o ‘DBFField‘ na ‘List<DBFField>‘
And o sistema remove todos os registros (‘DBFRecord‘) do ‘DBFFile‘ para evitar inconsistências
And o sistema atualiza o ‘DBFFile‘ e chama a validação (‘DBFValidation‘)
And o sistema retorna o ‘DBFFile‘ atualizado

Scenario: Falha ao modificar um campo inexistente Given que o ‘DBFFile‘ não contém um campo chamado ‘"Altura"‘ na ‘List<DBFField>‘
When o usuário tenta modificar o campo ‘"Altura"‘
Then o sistema deve lançar um erro ‘"Erro: O campo ’Altura’ não existe no .DBF"‘

And o sistema não deve modificar a ‘List<DBFField>‘

Scenario: Falha ao modificar um campo para um nome inválido Given que o usuário solicita a alteração do campo ‘"Peso"‘ para ‘"PesoMuitoGrandeDemais"‘
When o sistema valida o novo nome do campo
Then o sistema deve lançar um erro ‘"Erro: O nome do campo não pode exceder 10 caracteres"‘
And o sistema não deve modificar a ‘List<DBFField>‘

Scenario: Falha ao modificar um campo para um tipo inválido Given que o usuário solicita a alteração do tipo do campo ‘"Altura"‘ para ‘"M"‘ (Memo)
When o sistema valida o novo tipo do campo
Then o sistema deve lançar um erro ‘"Erro: O tipo ’M’ não é suportado pelo formato dBase III"‘
And o sistema não deve modificar a ‘List<DBFField>‘

1.14.3 Scenario: Remover um campo (‘DBFField‘) do ‘.DBF‘

Given que o ‘DBFFile‘ contém um campo chamado ‘"Idade"‘ na ‘List<DBFField>‘
When o usuário solicita a remoção do campo ‘"Idade"‘
Then o sistema deve validar que o campo existe na ‘List<DBFField>‘
And o sistema deve remover o campo ‘"Idade"‘ da ‘List<DBFField>‘
And o sistema remove todos os registros (‘DBFRecord‘) do ‘DBFFile‘ para evitar inconsistências
And o sistema atualiza o ‘DBFFile‘ e chama a validação (‘DBFValidation‘)
And o sistema retorna o ‘DBFFile‘ atualizado

Scenario: Falha ao remover um campo inexistente Given que o ‘DBFFile‘ não contém um campo chamado ‘"Altura"‘ na ‘List<DBFField>‘
When o usuário tenta remover o campo ‘"Altura"‘
Then o sistema deve lançar um erro ‘"Erro: O campo ’Altura’ não existe no .DBF"‘
And o sistema não deve modificar a ‘List<DBFField>‘

Scenario: Falha ao remover todos os campos do ‘.DBF‘ Given que o ‘DBFFile‘ contém apenas um campo na ‘List<DBFField>‘
When o usuário tenta remover esse único campo
Then o sistema deve lançar um erro ‘"Erro: O .DBF deve conter pelo menos um campo"‘

And o sistema não deve modificar a ‘List<DBFField>‘

1.15 Definição do BOUNDED CONTEXT: DBFRecordManagement

O ‘DBFRecordManagement‘ é o contexto responsável pela manipulação de registros (‘DBFRecord‘) dentro de um arquivo ‘.DBF‘. Esse contexto garante que as operações de inserção, modificação, remoção e recuperação de registros sejam realizadas de maneira segura e compatível com as restrições impostas pelo formato ‘dBase III (0x03)‘.

No formato ‘.DBF‘, os registros armazenam valores que seguem a estrutura definida pelos campos (‘DBFField‘). Cada registro (‘DBFRecord‘) deve obedecer ao tamanho fixo de registro (‘recordSize‘) e manter consistência com os metadados definidos no cabeçalho (‘DBFHeader‘). Além disso, a remoção de registros é realizada de forma lógica, marcando o registro como excluído (‘0x2A‘), sem remover fisicamente os dados do arquivo, o que permite futuras recuperações ou reuso de espaço.

Dada a necessidade de manter a integridade estrutural do ‘.DBF‘, todas as operações realizadas sobre os registros são validadas antes de serem aplicadas. O ‘DBFRecordManagement‘ funciona em conjunto com ‘DBFFileDefinition‘ para garantir que as inserções respeitem a estrutura do arquivo e com ‘DBFValidation‘ para assegurar que todas as modificações estejam dentro das regras do formato ‘.DBF‘.

1.15.1 Como Esse Contexto Será Implementado?

O ‘DBFRecordManagement‘ será modelado como um conjunto de operações sobre a estrutura já definida pelo ‘DBFFile‘. Ele não modifica a estrutura dos campos (‘DBFField‘), mas permite manipular os registros (‘DBFRecord‘) de forma segura e eficiente.

Esse contexto se baseia nos seguintes princípios:

- Garantir que cada ‘DBFRecord‘ siga fielmente a estrutura dos campos (‘DBFField‘).
- Manter a integridade e alinhamento dos registros no arquivo ‘.DBF‘.
- Controlar a remoção lógica (‘0x2A‘) e a recuperação de registros.
- Assegurar que as operações de inserção e modificação respeitem os limites do formato ‘dBase III (0x03)‘.

Para isso, o contexto opera diretamente sobre a ‘List<DBFRecord>‘ armazenada no ‘DBFFile‘, garantindo que os registros sejam sempre gerenciados de acordo com as definições estabelecidas pelo ‘DBFFileDefinition‘.

1.15.2 Organização do DBFRecordManagement

Esse contexto gerencia exclusivamente os registros (‘DBFRecord‘) de um ‘.DBF‘. Ele **não altera a estrutura do arquivo** e **não modifica os campos (‘DBFField‘)**, garantindo que as operações realizadas sobre os registros sejam independentes da definição estrutural do arquivo.

Esse contexto se divide nas seguintes responsabilidades:

- **Inserção de registros:** Adicionar novos registros (‘DBFRecord‘) ao ‘.DBF‘, garantindo que os valores respeitem os tipos de dados dos campos (‘DBFField‘).
- **Edição de registros:** Modificar valores dentro de registros existentes, assegurando que as mudanças estejam dentro dos limites da estrutura do ‘.DBF‘.
- **Exclusão lógica de registros:** Remover registros por meio da marcação ‘0x2A‘, sem excluir os dados fisicamente do arquivo.
- **Recuperação de registros:** Reverter a exclusão lógica e restaurar registros previamente removidos.
- **Consulta e filtragem de registros:** Permitir buscas dentro do ‘.DBF‘ com base em critérios específicos.
- **Remoção permanente de registros excluídos:** Eliminar registros marcados como removidos (‘0x2A‘) e reorganizar o espaço disponível no ‘.DBF‘.

1.15.3 Quais São as Responsabilidades Deste Contexto?

- Garantir que todos os registros (‘DBFRecord‘) sigam a estrutura definida pelos campos (‘DBFField‘).
- Manter a integridade dos registros dentro do ‘.DBF‘.
- Controlar o número total de registros (‘recordCount‘) no cabeçalho (‘DBFHeader‘).
- Assegurar que as operações sobre registros não comprometam a compatibilidade do arquivo com sistemas externos.
- Fornecer mecanismos seguros para recuperação e remoção definitiva de registros.

1.15.4 O Que Esse Contexto NÃO Faz?

Embora o ‘DBFRecordManagement‘ seja essencial para a manipulação de registros dentro do ‘.DBF‘, algumas responsabilidades são delegadas a outros contextos para garantir a modularidade do sistema:

Esse contexto não:

- Modifica a estrutura do ‘.DBF‘ (campos ‘DBFField‘). Essa responsabilidade pertence ao ‘DBFFieldManagement‘.
- Garante a validade da estrutura do arquivo. Essa função é do ‘DBFValidation‘.
- Realiza a exportação do ‘.DBF‘ para um arquivo binário. A exportação é tratada pelo ‘DBFExport‘.

1.15.5 Relação do DBFRecordManagement com Outros Contextos

Relação com ‘DBFFileDefinition‘
Esse contexto depende da estrutura definida pelo ‘DBFFileDefinition‘ para garantir que as operações sobre registros sejam consistentes com a definição dos campos (‘DBFField‘). Ele não pode adicionar registros (‘DBFRecord‘) a um ‘.DBF‘ sem antes validar que a estrutura está correta.

Relação com ‘DBFFieldManagement‘
Como a estrutura dos registros depende dos campos (‘DBFField‘), qualquer alteração na estrutura pode invalidar os registros existentes. Se um campo for modificado, todos os registros precisam ser removidos e recriados para manter a compatibilidade.

Relação com ‘DBFValidation‘
Antes de realizar qualquer modificação nos registros (‘DBFRecord‘), esse contexto verifica a validade da estrutura do arquivo através do ‘DBFValidation‘. Isso impede que registros sejam inseridos ou alterados de maneira incorreta.

Relação com ‘DBFExport‘
Uma vez que os registros tenham sido gerenciados corretamente, o ‘DBFExport‘ é responsável por transformar essa estrutura interna em um arquivo ‘.DBF‘ binário válido para uso externo.

1.15.6 Scenario: Criar um novo DBFRecord com valores válidos

Given um ‘DBFFile‘ existente com pelo menos um ‘DBFField‘ definido
And os valores a serem inseridos seguem a estrutura e os tipos de dados dos ‘DBFField‘
When o usuário solicita a criação de um novo ‘DBFRecord‘

Then o sistema deve iniciar um novo ‘DBFRecord‘
And para cada ‘DBFField‘ existente:

- O sistema verifica se há um valor correspondente
- O sistema valida se o tipo de dado está correto
- O sistema adiciona o valor ao ‘DBFRecord‘

And o sistema adiciona o ‘DBFRecord‘ à ‘List<DBFRecord>‘ do ‘DBFFile‘
And o sistema atualiza o ‘recordCount‘ no ‘DBFHeader‘
And o sistema retorna o ‘DBFRecord‘ criado

1.15.7 Scenario: Falha ao criar um DBFRecord com valores inválidos

Given um ‘DBFFile‘ com ‘DBFField‘ definidos
And o usuário fornece um valor incompatível com o tipo de dado do ‘DBFField‘
When o sistema valida os valores inseridos
Then o sistema detecta a incompatibilidade de tipo
And o sistema exibe um erro ‘"Erro: O campo ’Idade’ espera um valor numérico, mas recebeu uma string."‘
And o sistema não deve criar o ‘DBFRecord‘

1.15.8 Scenario: Falha ao criar um DBFRecord que excede ‘recordSize‘

Given um ‘DBFFile‘ onde a soma dos tamanhos dos campos (‘DBFField.size‘) já atinge o limite permitido
And o usuário tenta adicionar um ‘DBFRecord‘ cujo tamanho total ultrapassa ‘recordSize‘
When o sistema calcula o tamanho total do registro
Then o sistema detecta que o tamanho total excede o limite
And o sistema exibe um erro ‘"Erro: O tamanho total dos registros excede o limite permitido pelo formato dBase III."‘
And o sistema não deve criar o ‘DBFRecord‘

1.15.9 Scenario: Atualizar um DBFRecord com valores válidos

Given um ‘DBFFile‘ existente contendo pelo menos um ‘DBFRecord‘
And os novos valores seguem a estrutura e os tipos de dados dos ‘DBFField‘
And o ‘DBFRecord‘ alvo está presente na ‘List<DBFRecord>‘
When o usuário solicita a atualização do ‘DBFRecord‘
Then o sistema localiza o ‘DBFRecord‘ pelo identificador único
And para cada ‘DBFField‘ no ‘DBFRecord‘:

- O sistema verifica se há um novo valor correspondente
- O sistema valida se o novo valor está correto para o tipo do ‘DBFField‘
- O sistema substitui o valor antigo pelo novo valor

And o sistema atualiza a ‘lastUpdated‘ no ‘DBFHeader‘
And o sistema retorna o ‘DBFRecord‘ atualizado

1.15.10 Scenario: Falha ao atualizar um DBFRecord inexistente

Given um ‘DBFFile‘ que não contém o ‘DBFRecord‘ especificado
When o usuário solicita a atualização do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro especificado não existe no arquivo DBF."‘
And o sistema não realiza nenhuma modificação

1.15.11 Scenario: Falha ao atualizar um DBFRecord com valores inválidos

Given um ‘DBFFile‘ contendo um ‘DBFRecord‘
And o usuário fornece um valor incompatível com o tipo do ‘DBFField‘
When o sistema valida os novos valores
Then o sistema detecta a incompatibilidade de tipo
And o sistema exibe um erro ‘"Erro: O campo ’DataNascimento’ espera uma data no formato YYYYMMDD, mas recebeu um valor inválido."‘
And o sistema não atualiza o ‘DBFRecord‘

1.15.12 Scenario: Falha ao atualizar um DBFRecord que ultrapassa ‘recordSize‘

Given um ‘DBFFile‘ onde os ‘DBFRecord‘ seguem um tamanho fixo definido por ‘recordSize‘
And o usuário tenta atualizar um ‘DBFRecord‘ com valores que excedem esse limite
When o sistema calcula o novo tamanho do ‘DBFRecord‘
Then o sistema detecta que o novo tamanho excede ‘recordSize‘
And o sistema exibe um erro ‘"Erro: O tamanho total do registro atualizado excede o limite permitido pelo formato dBase III."‘
And o sistema não realiza nenhuma atualização

1.15.13 Scenario: Remover um DBFRecord existente

Given um ‘DBFFile‘ contendo pelo menos um ‘DBFRecord‘
And o ‘DBFRecord‘ alvo está presente na ‘List<DBFRecord>‘
When o usuário solicita a remoção do ‘DBFRecord‘
Then o sistema localiza o ‘DBFRecord‘ pelo identificador único
And o sistema marca o ‘DBFRecord‘ como excluído (‘isDeleted = true‘)
And o sistema atualiza ‘recordCount‘ no ‘DBFHeader‘
And o sistema atualiza ‘lastUpdated‘ no ‘DBFHeader‘
And o sistema retorna o ‘DBFFile‘ atualizado

1.15.14 Scenario: Falha ao remover um DBFRecord inexistente

Given um ‘DBFFile‘ que não contém o ‘DBFRecord‘ especificado
When o usuário solicita a remoção do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro especificado não existe no arquivo DBF."‘
And o sistema não realiza nenhuma modificação

1.15.15 Scenario: Falha ao remover um DBFRecord em um DBF vazio

Given um ‘DBFFile‘ sem registros (‘List<DBFRecord>‘ vazia)
When o usuário solicita a remoção de um ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: Não há registros disponíveis para remoção."‘
And o sistema não realiza nenhuma modificação

1.15.16 Scenario: Remover todos os registros de um DBFFile

Given um ‘DBFFile‘ contendo múltiplos ‘DBFRecord‘
When o usuário solicita a remoção de todos os registros
Then o sistema percorre a ‘List<DBFRecord>‘ e marca todos como excluídos (‘isDeleted = true‘)
And o sistema redefine ‘recordCount = 0‘ no ‘DBFHeader‘
And o sistema atualiza ‘lastUpdated‘ no ‘DBFHeader‘
And o sistema retorna o ‘DBFFile‘ atualizado

1.15.17 Scenario: Buscar um DBFRecord existente pelo identificador

Given um ‘DBFFile‘ contendo pelo menos um ‘DBFRecord‘
And o ‘DBFRecord‘ alvo está presente na ‘List<DBFRecord>‘
When o usuário solicita a busca pelo identificador único do ‘DBFRecord‘

Then o sistema percorre a ‘List<DBFRecord>‘ e localiza o ‘DBFRecord‘ correspondente
And o sistema retorna o ‘DBFRecord‘ com seus valores

1.15.18 Scenario: Falha ao buscar um DBFRecord inexistente

Given um ‘DBFFile‘ que não contém o ‘DBFRecord‘ especificado
When o usuário solicita a busca pelo identificador do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro especificado não existe no arquivo DBF."‘
And o sistema não retorna nenhum dado

1.15.19 Scenario: Buscar um DBFRecord marcado como excluído

Given um ‘DBFFile‘ contendo um ‘DBFRecord‘ com ‘isDeleted = true‘
When o usuário solicita a busca pelo identificador do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro especificado foi excluído e não pode ser acessado."‘
And o sistema não retorna nenhum dado

1.15.20 Scenario: Buscar todos os DBFRecords de um DBFFile

Given um ‘DBFFile‘ contendo múltiplos ‘DBFRecord‘
When o usuário solicita a busca de todos os registros
Then o sistema retorna uma ‘List<DBFRecord>‘ contendo todos os registros não excluídos
And o sistema ignora os ‘DBFRecord‘ onde ‘isDeleted = true‘

1.15.21 Scenario: Falha ao buscar registros em um DBFFile vazio

Given um ‘DBFFile‘ sem registros (‘List<DBFRecord>‘ vazia)
When o usuário solicita a busca de todos os registros
Then o sistema exibe um erro ‘"Erro: Não há registros disponíveis para consulta."‘
And o sistema não retorna nenhum dado

1.15.22 Scenario: Restaurar um DBFRecord excluído

Given um ‘DBFFile‘ contendo um ‘DBFRecord‘ marcado como excluído (‘isDeleted = true‘)
When o usuário solicita a restauração do ‘DBFRecord‘
Then o sistema redefine ‘isDeleted = false‘ no ‘DBFRecord‘

And o sistema atualiza o ‘lastUpdated‘ do ‘DBFHeader‘ com a data atual
And o sistema confirma a restauração com sucesso

1.15.23 Scenario: Falha ao restaurar um DBFRecord inexistente

Given um ‘DBFFile‘ sem o ‘DBFRecord‘ especificado
When o usuário solicita a restauração do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro especificado não existe no arquivo DBF."‘
And o sistema não realiza nenhuma restauração

1.15.24 Scenario: Falha ao restaurar um DBFRecord que não está excluído

Given um ‘DBFFile‘ contendo um ‘DBFRecord‘ com ‘isDeleted = false‘
When o usuário solicita a restauração do ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: O registro não está marcado como excluído."‘
And o sistema não realiza nenhuma restauração

1.15.25 Scenario: Falha ao restaurar registros quando o DBFFile está vazio

Given um ‘DBFFile‘ sem nenhum ‘DBFRecord‘ armazenado
When o usuário solicita a restauração de um ‘DBFRecord‘
Then o sistema exibe um erro ‘"Erro: Não há registros excluídos no arquivo DBF para serem restaurados."‘
And o sistema não realiza nenhuma restauração
Given um ‘DBFFile‘ contendo um ‘DBFRecord‘ válido
And o ‘DBFRecord‘ alvo está presente na ‘List<DBFRecord>‘
And os novos valores seguem a estrutura definida em ‘List<DBFField>‘
When o usuário solicita a atualização do ‘DBFRecord‘ com novos valores
Then o sistema valida que os valores são compatíveis com os tipos definidos em ‘DBFField‘
And o sistema atualiza os valores do ‘DBFRecord‘
And o ‘lastUpdated‘ do ‘DBFHeader‘ é atualizado com a data atual
And o sistema confirma a atualização com sucesso