

# PROJETO INTEGRADOR

Cristian, Luis Felipe e Vitor Correia

# ESTRUTURA DE DADOS

Benchmark

Critérios	Código grupo	Código aberto
Linguagem	TypeScript	C
Estrutura	Min-Heap	Lista Encadeada Simples
Lógica de priorização	Baseada nos níveis de prioridade SUS. Operações de inserção e extração do prioritário são $O(\log n)$ .	A lógica para manter a ordem ou encontrar o próximo prioritário pode ser $O(n)$ .
Complexidade	<ul style="list-style-type: none"> <li>→ Adicionar Paciente (com prioridade): <math>O(\log n)</math> - Muito eficiente. A nova prioridade <b>sempre sobe para sua posição correta no heap.</b></li> <li>→ Chamar Próximo Paciente (extrair o de maior prioridade): <math>O(\log n)</math> - Muito eficiente. O elemento raiz é removido e o heap é reorganizado.</li> <li>→ Ver Próximo Paciente (sem remover): <math>O(1)</math> - Simplesmente olha o elemento raiz.</li> </ul>	<ul style="list-style-type: none"> <li>→ Adicionar Paciente: <math>O(n)</math> no pior caso, pois pode ser necessário percorrer a lista para encontrar a posição correta de inserção para manter a ordem.</li> <li>→ Chamar Próximo Paciente do início : <math>O(1)</math>.</li> <li>→ A complexidade da inserção torna-se um problema.</li> </ul>

# PROGRAMAÇÃO ORIENTADA A OBJETOS

## Análise do código

```
abstract class Pessoa {
    constructor(public nome: string, public idade: number, public cpf: string) {}
    abstract apresentar(): string;
}

type NivelPrioridadeSUS = 'vermelho' | 'laranja' | 'amarelo' | 'verde' | 'azul';
const NIVEIS_PRIORIDADE_VALIDOS: NivelPrioridadeSUS[] = ['vermelho', 'laranja', 'amarelo', 'verde', 'azul'];

class Paciente extends Pessoa {
    public prioridade: NivelPrioridadeSUS;
    public descricaoPrioridade: string;

    constructor(
        nome: string,
        idade: number,
        cpf: string,
        public sintoma: string,
        nivelPrioridadeEntrada: NivelPrioridadeSUS
    ) {
        super(nome, idade, cpf);
        this.prioridade = nivelPrioridadeEntrada;
        this.descricaoPrioridade = this.obterDescricaoPrioridade(nivelPrioridadeEntrada);
    }

    private obterDescricaoPrioridade(prioridade: NivelPrioridadeSUS): string {
        switch (prioridade) {
            case 'vermelho': return 'Emergência (Atendimento Imediato)';
            case 'laranja': return 'Muito Urgente (Atendimento em até 10 min)';
            case 'amarelo': return 'Urgente (Atendimento em até 60 min)';
            case 'verde': return 'Pouco Urgente (Atendimento em até 120 min)';
            case 'azul': return 'Não Urgente (Atendimento em até 240 min ou encaminhamento)';
            default: return 'Indefinida';
        }
    }

    apresentar(): string {
        return `Paciente: ${this.nome}, Idade: ${this.idade}, Sintoma: ${this.sintoma}, Classificação de Risco: ${this.prioridade.toUpperCase()} (${this.descricaoPrioridade})`;
    }
}

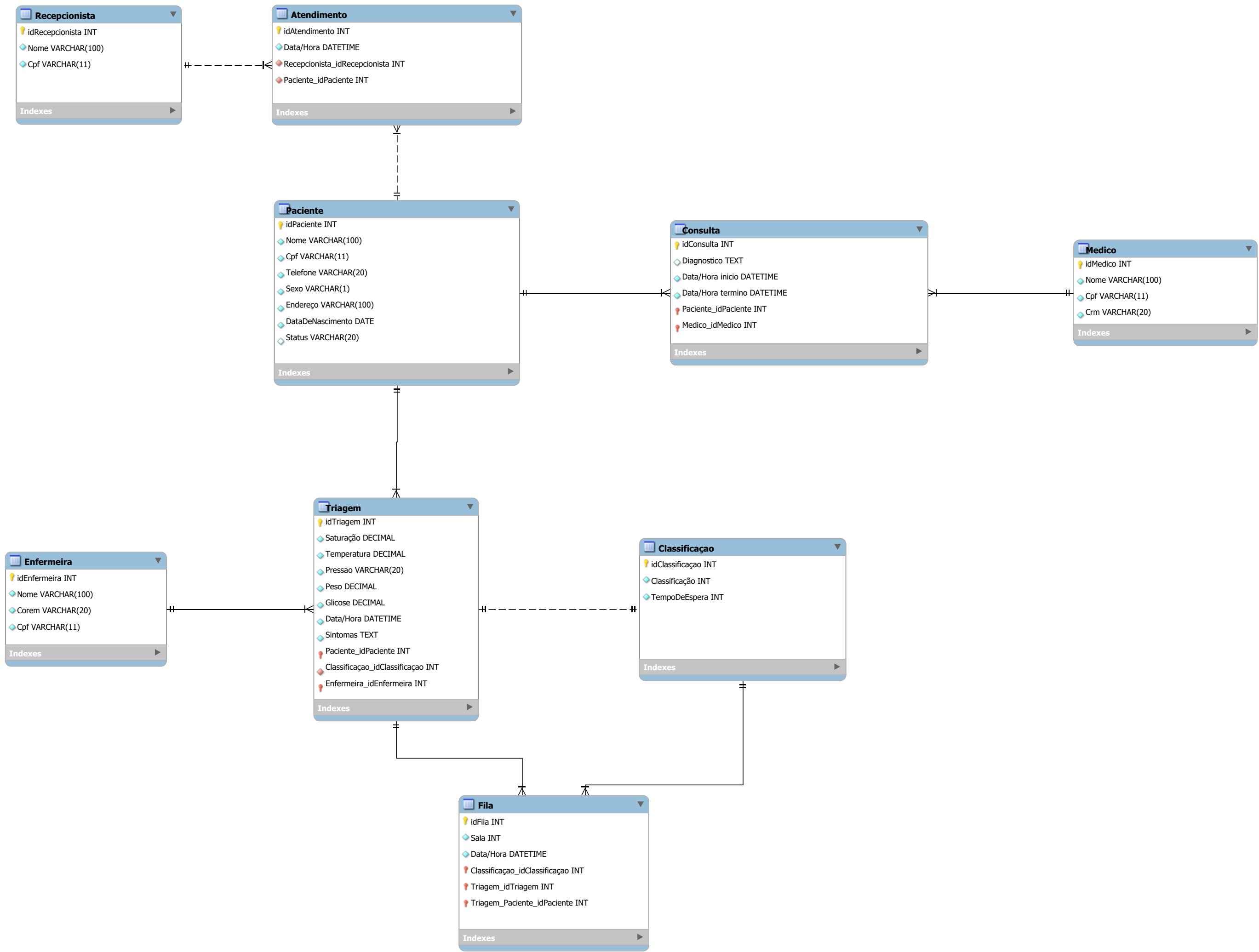
class Medico extends Pessoa {
    constructor(
        nome: string,
        idade: number,
        cpf: string,
        public especialidade: string,
        public crm: string
    ) {
        super(nome, idade, cpf);
    }

    apresentar(): string { return `Médico(a): ${this.nome}, Especialidade: ${this.especialidade}, CRM: ${this.crm}`; }

    atenderPaciente(paciente: Paciente): string {
        return `O(A) Médico(a) ${this.nome} (${this.especialidade}) está iniciando o atendimento ao(à) paciente ${paciente.nome} (Risco: ${paciente.prioridade.toUpperCase()}) com sintoma de "${paciente.sintoma}".`;
    }
}
```

# BANCO DE DADOS

## Diagrama relacional



# OBRIGADO!