CAPÍTULO 8

REVISÃO DE TECNOLOGIAS PARA CONTROLE AUTOMATIZADO DE MÁQUINAS DE COSTURA: PADRÕES DE INTERFACE, LINGUAGENS E MODELOS DE DADOS

https://doi.org/10.22533/at.ed.5832531038

Data de submissão: 06/06/2025 *Data de aceite: 16/06/2025*

Kleiton de Camargo
Universidade Federal do Paraná, Departamento de Engenharia Elétrica
Curitiba – Paraná
Lattes – http://lattes.cnpq.br/3385378391415154

Bruno Knevitz Hammerschmitt
Universidade Federal do Paraná, Departamento de Engenharia Elétrica
Curitiba – Paraná
Lattes – http://lattes.cnpq.br/4865207592578956

Marcos Vinicio Haas Rambo
Universidade Federal do Paraná, Departamento de Engenharia Elétrica
Curitiba – Paraná
Lattes – http://lattes.cnpq.br/1121386710402716

RESUMO: Este artigo apresenta uma análise integrada sobre os principais elementos envolvidos no desenvolvimento de um sistema de automação para a manufatura têxtil. A abordagem contempla as linguagens de programação e os padrões arquiteturais utilizados em aplicações Windows. O estudo compara as linguagens de marcação HTML e XAML, destacando suas aplicações, funcionalidades e contextos de execução. Além disso, discorre sobre o uso da linguagem C#, a estrutura do Windows Presentation Foundation e o padrão Model-View-Controller como base para a construção de interfaces modernas. Por fim, o trabalho apresenta os conceitos fundamentais de banco de dados relacionais e não relacionais, suas principais diferenças e aplicações práticas. A proposta contribui para a compreensão de conceitos de sistemas computacionais aplicados à automação de processos.

PALAVRAS-CHAVE: Costura; Automação; C#; Windows Presentation Foundation; Model-View-Controller; Banco de Dados.

REVIEW OF TECHNOLOGIES FOR AUTOMATED CONTROL OF SEWING MACHINES: INTERFACE STANDARDS, LANGUAGES AND DATA MODELS

ABSTRACT: This article presents an integrated analysis of the main elements involved in the development of an automation system for textile production. The approach includes the programming languages and architectural patterns used in Windows applications. The study compares the HTML

and XAML markup languages, highlighting their applications, functionalities and execution contexts. In addition, I discuss the use of the C# language, the Windows Presentation Foundation structure and the Model-View-Controller pattern as a basis for building modern interfaces. Finally, the work presents the fundamental concepts of relational and non-relational databases, their main differences and applications. The proposal contributes to the understanding of concepts of computer systems applied to process automation.
KEYWORDS: Sewing; Automation; C#; Windows Presentation Foundation; Model-View-Controller; Database.

1. INTRODUÇÃO

Com os avanços tecnológicos das últimas décadas, a automação de processos passou a ocupar papel central em diversos setores, desde linhas de produção industriais até tarefas domésticas. No contexto da indústria têxtil, a modernização de equipamentos tradicionais, como as máquinas de costura, tem contribuído significativamente para o aumento da eficiência, da padronização e da produtividade. Aliar componentes mecânicos a sistemas computacionais tornou-se uma estratégia essencial para atender às exigências crescentes de precisão e flexibilidade na produção.

Paralelamente, o desenvolvimento de interfaces gráficas mais intuitivas e responsivas tem possibilitado maior controle e facilidade de operação por parte dos usuários. Nesse cenário, as plataformas de desenvolvimento baseadas no sistema Windows, como o Windows Presentation Foundation (WPF), em conjunto com linguagens modernas como C# e XAML, oferecem recursos avançados para a construção de aplicações robustas e integradas.

Este artigo apresenta conceitos que podem ser aplicados para a automação de processos industriais, descrevendo a sua importância para a manufatura têxtil. O estudo contempla os fundamentos técnicos dos equipamentos, a estruturação de interfaces utilizando HTML e XAML, a aplicação do padrão arquitetural Model-View-Controller (MVC) e o papel dos bancos de dados relacionais e não relacionais na sustentação das funcionalidades do sistema. A proposta visa proporcionar uma visão integrada e aplicada sobre o desenvolvimento de soluções computacionais, boas práticas e decisões técnicas que impactam diretamente na eficiência, na usabilidade e na manutenção de sistemas automatizados.

2. DESENVOLVIMENTO

2.1. MANUFATURA TÊXTIL

A definição genérica de máquina de costura segundo Grace Rogers Cooper em seu livro “The Sewing Machine: its invention and development” é um dispositivo mecânico ou eletromecânico projetado para unir tecidos usando um fio de costura (Cooper, 1976). Esse equipamento automatiza o processo de costura, tornando-o mais rápido e consistente em

comparação com a costura manual, e pode ser usado para diferentes materiais, como tecidos, couro e outros materiais sintéticos.

Existem basicamente dois tipos de máquinas de costura disponíveis: industriais e domésticas. As máquinas de costura industriais são maiores, mais rápidas e mais variadas em tamanho, custo, aparência e tarefa. Uma máquina de costura industrial pode lidar com trabalhos de costura pesados. As máquinas industriais, ao contrário das máquinas domésticas, executam uma única tarefa dedicada e são capazes de uso contínuo por longos períodos. Elas têm peças móveis maiores e motores maiores classificados para operação contínua. Peças para diferentes máquinas industriais, como motores, pés de costura e bobinas, podem ser intercambiáveis, mas nem sempre é assim.

Os motores das máquinas industriais, como a maioria de seus componentes, são geralmente montados na parte inferior da mesa, enquanto as máquinas domésticas têm seus motores OEM montados dentro da máquina. Existem dois tipos diferentes de motor disponíveis para máquinas industriais: um servo motor (que consome menos eletricidade e é silencioso quando não está em uso) e o motor de embreagem mais tradicional (que está sempre girando, mesmo quando não está em uso). Um motor de embreagem está sempre funcionando e fazendo barulho quando está conectado à eletricidade. A operação constante garante consistência e velocidade. O servo motor usa menos eletricidade do que um motor de embreagem e não emite nenhum som, a menos que o operador pise no pedal da máquina, mas não pode suportar o mesmo tipo de uso que um motor de embreagem.

Na manufatura têxtil, especialmente em processos de costura especializados com requisitos de segurança, falhas operacionais e humanas são causas recorrentes de desperdícios, retrabalhos e comprometimento da qualidade final dos produtos. Pequenos erros na sequência de costura podem resultar em peças descartadas, prejuízos econômicos e, em casos mais críticos, risco à integridade de componentes cuja segurança estrutural depende da precisão do processo da costura, como ocorre em setores automotivo, aeroespacial e médico-hospitalar.

A ausência de controle rigoroso sobre a execução de cada etapa da costura compromete a rastreabilidade e dificulta a padronização da produção. Em ambientes industriais com alta demanda, a repetitividade da atividade e a dependência de ações humanas sem assistência técnica adequada tornam o sistema suscetível a falhas que poderiam ser evitadas com mecanismos de prevenção de erros.

Neste contexto, o desenvolvimento de um sistema visual de poka-yoke para máquinas de costura industriais, com foco na prevenção de falhas operacionais, pode garantir a conformidade do processo e aumento da eficiência produtiva. Linguagens de programação para ambientes WEB e Windows, com frontend amigáveis e integrados com bancos de dados com registros do processo (operação, falhas, imagens e dados do usuário), são uma boa abordagem, pois garantem a qualidade e repetibilidade do processo fabril e sua rastreabilidade, viabilizando a implantação de processos de melhoria contínua.

2.2. APLICATIVO WINDOWS

2.2.1. Linguagem HTML

De acordo com Mozilla (2025), HTML (Linguagem de Marcação de Hipertexto, do inglês HyperText Markup Language) é uma linguagem usada para definir a estrutura de uma página web, atribuindo também significado ao seu conteúdo, sendo o bloco fundamental da internet (Mozilla, 2022). A Mozilla (2022) também define Hipertexto como links que conectam diferentes páginas da web, enquanto marcação ou tags se refere à forma como o navegador entende o que deve ser exibido e como. Um elemento é o conjunto formado por uma tag, seus atributos, valores e filhos.

As tags são componentes essenciais do HTML, pois com elas a estrutura da página é construída. Sua sintaxe é composta pelo nome da tag desejada, envolto por parênteses angulares (<>). Além disso, toda tag que for aberta deve ser fechada, seguindo a mesma estrutura, mas com uma barra antes do nome da tag.

2.2.2. Linguagem XAML

De acordo com a Microsoft (2023), XAML é uma linguagem de marcação declarativa que, no contexto do modelo de programação .NET, facilita a criação de interfaces de usuário para aplicativos .NET (Microsoft, 2023a). Com o XAML, você pode definir elementos visuais da interface diretamente na marcação, permitindo a separação entre a interface do usuário e a lógica de execução por meio de arquivos code-behind, que são conectados à marcação usando definições de classes parciais.

Ao contrário de muitas outras linguagens de marcação, que são normalmente interpretadas sem um vínculo direto com um sistema de tipos, o XAML permite a instanciação direta de objetos com base em um conjunto específico de tipos definidos em assemblies. Isso possibilita um fluxo de trabalho colaborativo, onde diferentes equipes podem trabalhar simultaneamente na interface do usuário e na lógica do aplicativo, utilizando ferramentas distintas e otimizando o processo de desenvolvimento.

2.2.2. HTML vs XAML: Comparação de Linguagens de Marcação

Pra fins de entendimento, HTML e XAML são linguagens de marcação utilizadas para definir a estrutura e a interface de diferentes tipos de aplicações. Embora compartilhem certas características, como a estrutura em árvore de tags, suas finalidades e ambientes de uso são distintos. Diante disto, será apresentado as algumas das principais características dessas linguagens de marcação.

2.2.3.1 Finalidade e Aplicação

O HTML é utilizado principalmente para estruturar o conteúdo de páginas web. Ele organiza textos, imagens, links e outros elementos visuais que são exibidos nos navegadores. O HTML serve como a base da web, trabalhando em conjunto com CSS (para estilização) e JavaScript (para interatividade) na criação de páginas dinâmicas e interativas (W3C, HTML Living Standard).

Por outro lado, o XAML é usado para criar interfaces de usuário em aplicações desktop, principalmente no Windows Presentation Foundation (WPF) e em outras tecnologias da Microsoft, como o Xamarin. O XAML facilita a separação entre a interface gráfica e o código lógico da aplicação, permitindo a definição de componentes como botões, caixas de texto e menus, entre outros, além de vincular eventos e lógica C# (Microsoft, XAML Overview, WPF).

2.2.3.2. Ambiente de Execução

O HTML é interpretado por navegadores web, como Chrome, Firefox e Edge, e depende de servidores web para distribuição. Ele é usado em conjunto com protocolos de rede, como HTTP e HTTPS, para criar aplicações web acessíveis em diferentes plataformas.

Por outro lado, o XAML é utilizado para criar aplicações desktop, que rodam localmente no sistema operacional Windows, através de frameworks como WPF ou UWP. Ele é ideal para criar interfaces gráficas complexas e nativas, com suporte à renderização gráfica acelerada por hardware e acesso direto aos componentes do sistema.

2.2.3.3. Interatividade e Lógica

O HTML, por si só, não lida diretamente com lógica e interatividade. Para isso, é necessário o uso de linguagens como JavaScript ou frameworks como React.js e Angular. O HTML define apenas a estrutura básica, enquanto JavaScript gerencia eventos como cliques, animações e outras interações.

No XAML, há uma integração direta com C# e outros frameworks .NET. Eventos e lógica de interação, como cliques em botões ou entradas de texto, são associados diretamente aos elementos da interface dentro do próprio XAML, permitindo uma interação mais rica e nativa com os controles e eventos do sistema operacional. Por exemplo, no WPF, é possível associar um botão diretamente a um evento C# no XAML, algo que em HTML exigiria JavaScript.

2.2.3.4. Estilos e Customização

O HTML utiliza o CSS (Cascading Style Sheets) para definir a aparência visual de uma página, controlando cores, tamanhos, espaçamentos e animações.

O XAML possui um sistema similar de estilização, com Styles e Templates, que permite a personalização da aparência dos controles e a criação de temas para a aplicação. A vantagem do XAML é sua integração profunda entre lógica e interface, suportando recursos como Data Binding e animações mais complexas (Microsoft, Styles and Templates in XAML).

2.2.3.5. Evolução e Suporte

O HTML é amplamente suportado por todas as plataformas e continua a evoluir como parte dos padrões da web, incorporando novas tecnologias como WebAssembly e Progressive Web Apps (PWAs).

O XAML, embora mais restrito ao ecossistema da Microsoft, é uma ferramenta poderosa para o desenvolvimento de aplicações de desktop e mobile nas plataformas WPF, UWP e Xamarin.Forms.

A principal diferença entre HTML e XAML está no contexto de uso e na funcionalidade que cada uma oferece. O HTML é projetado para a web, sendo utilizado para estruturar e exibir páginas web interativas, enquanto o XAML é voltado para aplicações desktop e mobile, permitindo a criação de interfaces gráficas ricas em aplicações baseadas no Windows.

2.2.4. Windows Presentation Foundation

De acordo com o artigo da Microsoft (2023):

> “ Windows Presentation Foundation (WPF), uma estrutura de interface do usuário que é independente de resolução e usa um mecanismo de renderização baseado em vetor, criado para aproveitar o hardware gráfico moderno. O WPF fornece um conjunto abrangente de recursos de desenvolvimento de aplicativos que incluem XAML (Extensible Application Markup Language), controles, associação de dados, layout, elementos gráficos 2D e 3D, animação, estilos, modelos, documentos, mídia, texto e tipografia. O WPF faz parte do .NET; portanto, você pode criar aplicativos que incorporam outros elementos da API .NET.” (Microsoft, 2023b).

A linguagem de marcação por trás da interface do WPF não é o HTML, mas sim o XAML (Extensible Application Markup Language). Embora ambos sejam linguagens de marcação, o XAML é utilizado especificamente para interfaces gráficas no WPF, enquanto o HTML é usado para estruturar páginas web. Diferente de uma interface web convencional, no WPF é possível associar eventos a cada componente da interface. Por exemplo, ao inserir um botão, é possível vincular diversos tipos de eventos a ele, como quando o mouse

passa sobre o botão, quando é clicado, ou quando é clicado e segurado, entre outros. Há uma ampla variedade de eventos disponíveis, o que facilita a programação de interações. A programação com WPF se assemelha, de certa forma, à programação de aplicativos no Android Studio, especialmente no que diz respeito à manipulação de interfaces gráficas e eventos.

2.2.5. Linguagem de programação C#

De acordo com Felipe Poter (2022), C# (pronunciado “C sharp”) é uma linguagem de programação moderna e versátil, desenvolvida e lançada em 2000 como parte da plataforma .NET (Poter, 2022). Criada por Anders Hejlsberg e sua equipe, a linguagem rapidamente se destacou como uma das mais populares para o desenvolvimento de aplicativos Windows, aplicativos web e soluções móveis. Com seu crescimento contínuo, C# se tornou uma escolha robusta e confiável em diversos cenários, desde pequenas aplicações a grandes sistemas empresariais.

Entre suas características mais marcantes de acordo com a Microsoft (2025), C# se destaca por (Microsoft, 2025):

- Orientação a Objetos: A linguagem é totalmente orientada a objetos, o que significa que tudo em C# gira em torno de objetos e classes, tornando o código modular, reutilizável e mais fácil de manter.
- Gerenciamento Automático de Memória: Com um coletor de lixo integrado, C# lida automaticamente com a alocação e liberação de memória, reduzindo a carga do desenvolvedor e minimizando problemas comuns de gerenciamento de memória, como vazamentos.
- Tipagem Forte: C# é fortemente tipada, o que significa que os tipos de dados são rigorosamente controlados pelo compilador, ajudando a evitar erros comuns e a garantir maior segurança e precisão no código.
- Foco em Segurança: Projetada com segurança em mente, C# inclui mecanismos internos para proteger dados e evitar vulnerabilidades, tornando-a uma excelente escolha para o desenvolvimento de aplicações em ambientes empresariais e críticos.

Para programar em C#, o Visual Studio é a IDE (Ambiente de Desenvolvimento Integrado) recomendada. Oferecida pela Microsoft, essa ferramenta oferece uma ampla gama de recursos, como edição avançada de código, depuração, design de interfaces gráficas, integração com bancos de dados e suporte a diversas tecnologias e frameworks. Isso faz do Visual Studio uma poderosa plataforma para desenvolvedores, facilitando o processo de desenvolvimento e aprimorando a produtividade.

Uma das grandes evoluções da linguagem é que, além de ser poderosa e flexível, C# é open-source, o que permite a contribuição e o uso livre por desenvolvedores de

todo o mundo. Isso significa que qualquer pessoa pode acessar, modificar e melhorar a linguagem e seus frameworks, promovendo maior transparência, inovação e colaboração na comunidade de desenvolvimento. A Microsoft abriu o código da linguagem e de seu ecossistema por meio da .NET Foundation, facilitando o uso de C# em diferentes plataformas, como Windows, macOS e Linux.

2.2.6. Padrão Arquitetural Model-View-Controller

O padrão de arquitetura Model-View-Controller (MVC) organiza o software em três componentes principais para promover a separação de responsabilidades e facilitar a manutenção (Gamma et al., 1994):

- Model (Modelo): Gerencia os dados e a lógica de negócio, independente da interface com o usuário. Esse componente lida com as regras de negócio e notifica os outros componentes quando há mudanças, mantendo a interface atualizada.
- View (Visão): Exibe os dados e responde às interações do usuário. Observa o modelo e apresenta as informações conforme o estado atual dos dados, assegurando que a interface seja sempre uma representação atualizada do modelo.
- Controller (Controlador): Atua como mediador entre a visão e o modelo, processando as ações do usuário, que são traduzidas em mudanças no modelo e atualizações na visão.
- O MVC permite a independência de desenvolvimento e teste de cada camada, aumentando a modularidade e a escalabilidade do software. É um padrão essencial para sistemas interativos e de fácil manutenção.

2.3. BANCO DE DADOS

2.3.1. Definição

De acordo com Silberschatz, Korth e Sudarshan (2019), um Sistema de Gerenciamento de Banco de Dados (SGBD) é uma coleção de dados inter-relacionados e um conjunto de programas para acessar esses dados. A coleção de dados, geralmente chamada de banco de dados, contém informações relevantes para uma organização. O principal objetivo de um SGBD é fornecer uma maneira conveniente e eficiente de armazenar e recuperar informações do banco de dados (Silberschatz; Korth; Sudarshan, 2019).

Os sistemas de banco de dados são projetados para gerenciar grandes volumes de informações. A gestão dos dados envolve tanto a definição de estruturas para o armazenamento de informações quanto a disponibilização de mecanismos para a

manipulação dessas informações. Além disso, o sistema de banco de dados deve garantir a segurança das informações armazenadas, mesmo em caso de falhas no sistema ou tentativas de acesso não autorizado. Quando os dados são compartilhados entre várias aplicações e usuários, como ilustrado na Figura 1, o sistema deve evitar possíveis resultados anômalos.

Dado o valor da informação para a maioria das organizações, cientistas da computação desenvolveram um vasto conjunto de conceitos e técnicas para o gerenciamento de dados. Silberschatz, Korth e Sudarshan (2019) citam também que os bancos de dados surgiram com o propósito de substituírem os velhos armários de arquivos, porém trabalhando de maneira semelhante. Os softwares de gerenciamento de banco de dados (SGDBs) surgiram anos mais tarde, devido a problemas com performance, precisão e confiabilidade na manipulação dos dados.

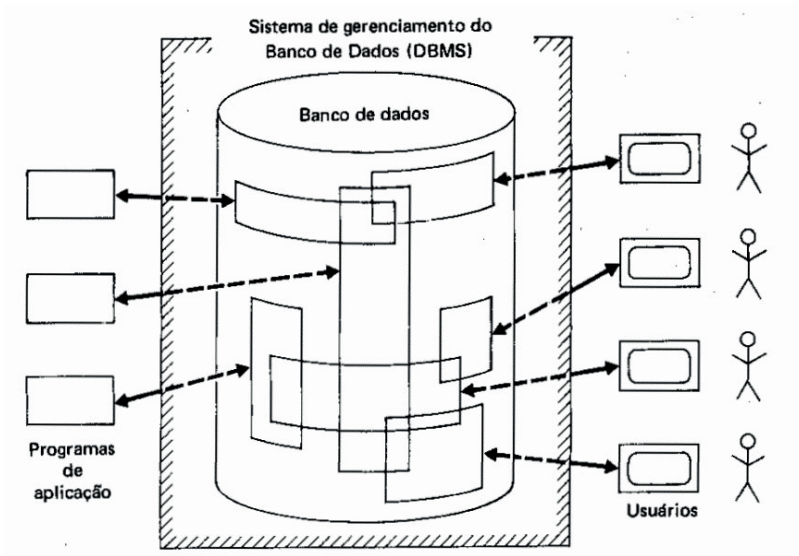

Figura 1 - BANCO DE DADOS E SUAS COMPONENTES

FONTE: (Date, 2003)

2.3.2. Abstração dos dados

Para garantir a eficiência no acesso aos dados e simplificar a interação dos usuários com o sistema, segundo Silberschatz, Korth e Sudarshan (2019), os bancos de dados são organizados em três níveis de abstração:

- Nível físico: Descreve como os dados são armazenados de forma detalhada em estruturas de baixo nível.
- Nível lógico: Mostra quais dados são armazenados e os relacionamentos entre eles, utilizando estruturas mais simples, ocultando a complexidade do nível físico.

- Nível de visão: Apresenta apenas partes específicas do banco de dados, adaptadas às necessidades dos usuários, tornando a interação mais simples e direta.

A Figura 2 ilustra esses níveis, os quais permitem que os usuários utilizem o sistema de forma eficiente sem lidar com a complexidade interna dos dados.

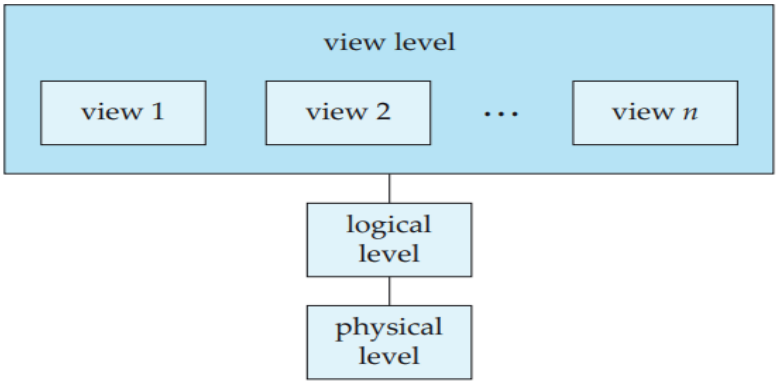

Figura 2 - NÍVEIS DE ABSTRAÇÃO BANCO DE DADOS

FONTE: (Silberschatz; Korth; Sudarshan, 2019)

2.3.3. Instâncias e esquemas

De acordo com Silberschatz, Korth e Sudarshan (2019), os dados dos Bancos de dados (DB) mudam o tempo todo, através da sua inserção ou exclusão. A coleção de informações armazenadas no banco de dados em um determinado momento é chamada de instância, enquanto o design do Banco de dados é chamado de esquema.

O conceito de esquema e instância pode ser entendido através da associação com um programa em linguagem de programação. Um esquema corresponde à declaração de uma variável em um programa. O valor da variável a cada momento é chamado de instância.

Os bancos de dados possuem diversos esquemas, divididos de acordo com suas camadas de abstração. O esquema físico descreve o banco em nível físico, assim como o esquema lógico o descreve em nível lógico. No nível de visualização existem diversos esquemas, sendo também chamados de subesquemas, que descrevem as diferentes visualizações do banco.

2.3.4. Modelo de dados

De acordo com Silberschatz, Korth e Sudarshan (2019), os modelos de dados são ferramentas conceituais para descrever os dados, seus relacionamentos, semânticas e restrições de consistência em um banco de dados. Eles são essenciais para projetar bancos de dados nos diferentes níveis (físico, lógico e de visão). Existem quatro categorias principais de modelos de dados:

- Modelo Relacional: Usa tabelas para representar dados e seus relacionamentos. Cada tabela possui colunas e registros. Este é o modelo mais usados atualmente.
- Modelo Entidade-Relacionamento (E-R): Focado em entidades e os relacionamentos entre elas. Amplamente utilizado no design de banco de dados.
- Modelo Baseado em Objetos: Inspirado em linguagens de programação orientadas a objetos, combinando métodos e encapsulamento com atributos de dados.
- Modelo Semiestruturado: Permite que dados do mesmo tipo tenham diferentes conjuntos de atributos, comumente representados em XML.

Esses modelos ajudam a estruturar e gerenciar dados de forma eficiente em sistemas de banco de dados modernos.

2.3.5. Banco de dados Relacional

O modelo relacional é amplamente utilizado em aplicações comerciais de processamento de dados devido à sua simplicidade, que facilita o trabalho dos programadores em comparação com modelos anteriores, como o modelo de rede e o modelo hierárquico.

2.3.5.1. Estrutura de Bancos de Dados Relacionais

Um banco de dados relacional é composto por uma coleção de tabelas. Cada tabela tem um nome único e contém linhas (ou tuplas) e colunas (ou atributos). Por exemplo, a tabela de instrutores contém as colunas ID, nome, departamento e salário. Cada linha da tabela representa um instrutor específico com essas informações. A tabela cursos armazena informações de cursos, como ID do curso, título, departamento e créditos. O conceito de relação em matemática está diretamente relacionado ao conceito de tabela no modelo relacional, e uma linha da tabela é chamada de tupla.

2.3.5.2. Domínios e Valores Atômicos

Cada coluna de uma tabela tem um domínio, que é o conjunto de valores permitidos para aquela coluna. Por exemplo, o domínio da coluna salário inclui todos os valores possíveis de salários. No modelo relacional, os domínios devem ser atômicos, ou seja, os valores são indivisíveis. Por exemplo, se armazenamos números de telefone como um

único valor, o domínio é considerado atômico. Mas se for dividido o número em partes (código de área e número local), ele deixaria de ser atômico.

2.3.5.2. Valores Nulos:

Um valor nulo representa a ausência ou o desconhecimento de um valor. Por exemplo, um instrutor pode não ter um número de telefone, então um valor nulo seria usado. Contudo, os valores nulos podem gerar desafios em operações de acesso e atualização no banco de dados. Esses conceitos são a base para o funcionamento de bancos de dados relacionais. Basicamente, os bancos de dados relacionais são estruturados em tabelas, conforme ilustrado na Figura 3.

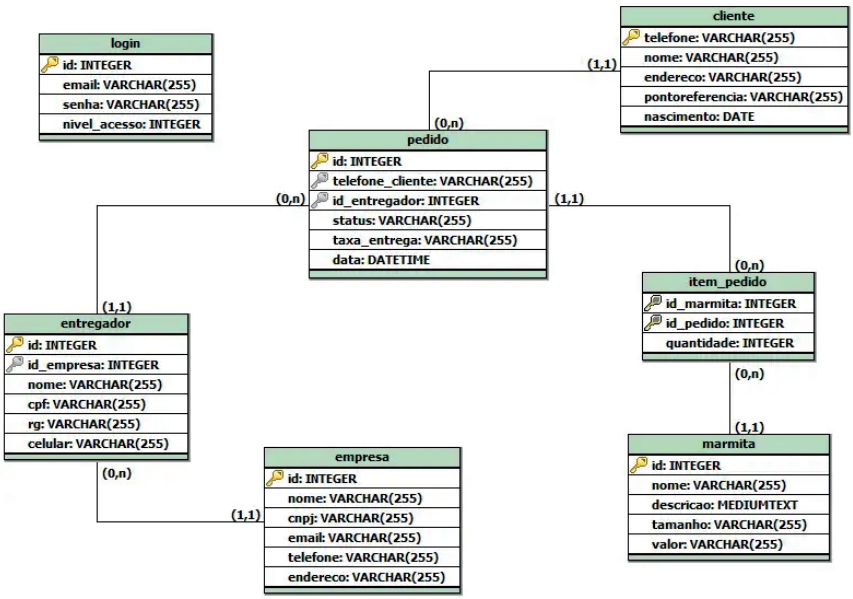

Figura 3 - Tabelas relacionais

FONTE: (Barroso, 2023)

As requisições de dados são feitas por meio de códigos específicos, chamados queries, que permitem buscar os dados armazenados nas tabelas de maneira lógica, relacionando os dados de uma com a outra e retornando as informações desejadas.

2.3.5.4. Exemplo de “Query”:

Com base no diagrama de tabelas fornecido, Figura 3, pode-se consultar o nome do cliente, o status do pedido, o nome da marmita e o nome do entregador para cada pedido feito, utilizando uma query SQL como o exemplo a seguir:

“SELECT cliente.nome AS NomeCliente,

```
pedido.status AS StatusPedido,
marmita.nome AS NomeMarmita,
entregador.nome AS NomeEntregador
FROM pedido
INNER JOIN cliente ON pedido.telefone_cliente = cliente.telefone
INNER JOIN item_pedido ON pedido.id = item_pedido.id_pedido
INNER JOIN marmita ON item_pedido.id_marmita = marmita.id
INNER JOIN entregador ON pedido.id_entregador = entregador.id;"
```

2.3.5.5. Resultado Esperado:

Neste exemplo, a requisição estabelece o relacionamento entre as tabelas de pedido, cliente, item do pedido, marmita e entregador, retornando informações combinadas de diversas tabelas inter-relacionadas, listadas na Tabela 1, conforme as chaves primárias e estrangeiras definidas no modelo de banco de dados relacional.

Cliente	Status_Pedido	Marmita	entregador
João Silva	Entregue	Marmita Média	Carlos Souza
Maria Lima	Pendente	Marmita Grande	José Santos

Tabela 1 - Resultado da requisição

2.3.5.6. Bancos de dados não-relacional

De acordo com Ramez e Shamkant (2017), os bancos de dados não relacionais, também chamados de "NoSQL", surgiram para atender a demandas específicas de aplicações que exigem grande volume de dados, alta velocidade de processamento e escalabilidade (Ramez; Shamkant, 2017). Enquanto os bancos de dados relacionais organizam dados em tabelas estruturadas, com um esquema rígido e pré-definido, os bancos de dados não relacionais oferecem maior flexibilidade ao armazenar dados em formatos variados, como documentos, grafos ou pares chave-valor.

2.3.5.6.1. Principais Tipos de Bancos de Dados Não Relacionais

Existem diversos tipos de bancos de dados, entretanto os tipos apresentados na Figura 4 são os mais utilizados:

- Bancos de Dados de Documentos: Armazenam dados em documentos (geralmente JSON, BSON ou XML), cada um com uma estrutura flexível. Exemplos: MongoDB e CouchDB.
- Bancos de Dados de Grafos: Modelam dados em nós (entidades) e arestas (relacionamentos), sendo ideais para representar relações complexas. Exemplos: Neo4j e ArangoDB.

- Bancos de Dados Chave-Valor: Funcionam como um dicionário, onde cada chave está associada a um valor específico. São eficientes em operações simples. Exemplos: Redis e DynamoDB.
- Bancos de Dados em Colunas: Armazenam dados por colunas ao invés de linhas, otimizando consultas de grandes volumes de dados. Exemplo: Cassandra.

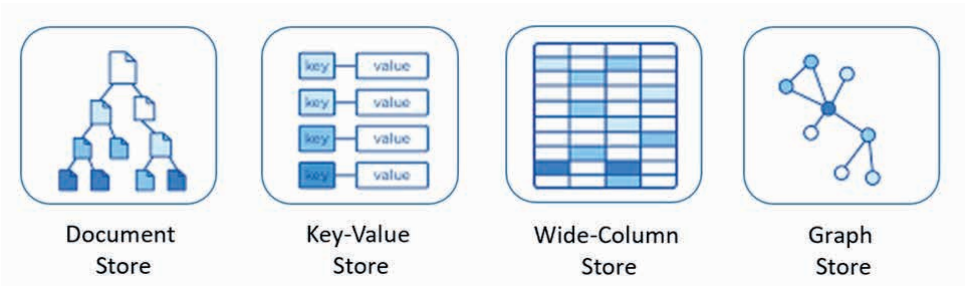

Figura 4 – Banco de dados não-relacional

FONTE: (Rocketseat, 2025)

2.3.7 Diferenças Principais Entre Bancos de Dados Relacionais e Não Relacionais

Da mesma forma, existem diversos tipos de bancos de dados relacionais e não relacionais, entretanto existem diferenças entre eles, as quais podem ser identificadas por:

- Estrutura de Dados: No relacional organizam dados em tabelas com um esquema rígido (modelo predefinido), garantindo integridade e consistência, enquanto não relacional usa estruturas flexíveis (documentos, chave-valor e grafos), permitindo a inclusão de dados sem alterações no esquema.
- Escalabilidade: No relacional a escalabilidade é vertical (aumento de capacidade em um único servidor), enquanto no não relacional a escalabilidade é horizontal (distribuição de dados e processamento em vários servidores).
- Flexibilidade: O relacional requer um esquema rígido para garantir a integridade dos dados, já o não relacional oferece flexibilidade, permitindo que dados diferentes coexistam sem restrições de formato.
- Consultas: O relacional utiliza a linguagem SQL (Structured Query Language) para consultas complexas. O Não Relacional utiliza APIs específicas ou linguagens próprias, como a de consulta do MongoDB.
- Consistência e Disponibilidade: O relacional se baseina no modelo ACID (Atomicidade, Consistência, Isolamento e Durabilidade) para garantir a integridade transacional. O não Relacional segue o modelo BASE (Basicamente Disponível, Estado Flexível, Consistência Eventual), priorizando disponibilidade e desempenho.

Assim, enquanto os bancos de dados relacionais são mais adequados para aplicações que exigem transações seguras e consistentes, os bancos não relacionais são ideais para sistemas distribuídos que processam grandes volumes de dados com estruturas flexíveis (Ramez; Shamkant, 2017).

A Figura 5 apresenta as principais diferenças entre os bancos de dados relacionais e os não relacionais.

Figura 5 – Relacional vs Não Relacional

FONTE: (Kondado, 2023)

3. CONCLUSÃO

Este trabalho proporcionou uma abordagem abrangente e aplicada do desenvolvimento de um sistema computacional para automação de processos industriais e modelagem de dados. A implementação da linguagem C# com o framework WPF, aliada ao uso de XAML para definição da interface gráfica, mostrou-se uma escolha assertiva ao possibilitar a criação de aplicações com alto nível de personalização, responsividade e manutenibilidade. A adoção do padrão Model-View-Controller (MVC) contribuiu significativamente para a separação de responsabilidades no código, otimizando a

escalabilidade e a organização da aplicação. Esses aspectos tornam a solução não apenas funcional, mas adaptável a futuras evoluções e melhorias.

Adicionalmente, o estudo sobre bancos de dados relacionais e não relacionais ressaltou o papel estratégico do armazenamento da informação, apresentando critérios técnicos para escolha da estrutura mais adequada de acordo com as exigências do sistema. A comparação permitiu refletir sobre desempenho, consistência, flexibilidade e escalabilidade, aspectos essenciais em soluções voltadas à automação.

Como contribuição prática, este trabalho serve como referência para o desenvolvimento de sistemas semelhantes, oferecendo diretrizes técnicas e arquiteturais que podem ser adaptadas a outras realidades industriais. Do ponto de vista acadêmico, o trabalho amplia o repertório de estudos que exploram a interdisciplinaridade, reforçando a importância de abordagens integradas para soluções complexas.

Dessa forma, conclui-se que o desenvolvimento de sistemas de automação baseados em plataformas modernas requer não apenas domínio técnico, mas também planejamento estruturado e visão sistêmica. O equilíbrio entre eficiência operacional, facilidade de uso e sustentabilidade tecnológica constitui o alicerce para inovações relevantes e duradouras no setor produtivo. Espera-se que este trabalho sirva de base para pesquisas de futuras acerca do assunto.

REFERÊNCIAS

BARROSO, Igor. **Banco de Dados Relacional: Fundamentos e Aplicações**. Disponível em: <https://www.dio.me/articles/banco-de-dados-relacional-fundamentos-e-aplicacoes>. Acesso em: 5 jun. 2025.

DATE, Chris J. **An Introduction to Database Systems**. 8ª ed. *[S.l.]*: Pearson, 2003.

GAMMA, Erich *et al.* **Design Patterns: Elements of Reusable Object-Oriented Software**. 1ª ed. *[S.l.]*: Addison-Wesley Professional, 1994.

POTER, Felipe. **Um pouco sobre C#**. Disponível em: <https://www.tabnews.com.br/FelipePoter/um-pouco-sobre-c>. Acesso em: 5 jun. 2025.

COOPER, Grace Rogers. **The Sewing Machine: its invention and development**. Disponível em: <https://www.sil.si.edu/DigitalCollections/hst/cooper/CF/view.cfm>. Acesso em: 5 jun. 2025.

KONDADO. **Banco de dados relacional vs o não relaciona**. Disponível em: <https://kondado.com.br/index.html>.

MICROSOFT. **Visão geral do XAML**. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-9.0>. Acesso em: 5 jun. 2025a.

MICROSOFT. **Visão geral do Windows Presentation Foundation**. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/overview/>. Acesso em: 5 jun. 2025b.

MICROSOFT. **Documentação da linguagem C#**. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/>. Acesso em: 5 jun. 2025.

MOZILLA. **HTML: Linguagem de Marcação de Hipertexto**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 5 jun. 2025.

RAMEZ, Elmasri; SHAMKANT, Navathe. **Fundamentals of database systems**. . Acesso em: 5 jun. 2025.

ROCKETSEAT. **Banco de Dados Não Relacional**. Disponível em: <https://www.rocketseat.com.br/blog>. Acesso em: 5 jun. 2025.

SILBERSCHATZ, Abraham; KORTH, Henry; SUDARSHAN, S. **ISE Database System Concepts**. 7ª ed. *[S.l.]*: McGraw-Hill Education, 2019.