UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THAILA ROANNI SCHMIDT CAVALHEIRO

UMA LINGUAGEM PARA A MODELAGEM LÓGICA DE BANCOS DE DADOS NO BRMODELO WEB

CHAPECÓ
2025

THAILA ROANNI SCHMIDT CAVALHEIRO

UMA LINGUAGEM PARA A MODELAGEM LÓGICA DE BANCOS DE DADOS NO BRMODELO WEB

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Geomar Andre Schreiner

CHAPECÓ
2025

THAILA ROANNI SCHMIDT CAVALHEIRO

UMA LINGUAGEM PARA A MODELAGEM LÓGICA DE BANCOS DE DADOS NO BRMODELO WEB

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação na Universidade Federal da Fronteira Sul.

Orientador: Prof. Geomar Andre Schreiner

Este Trabalho de Conclusão de Curso foi avaliado e aprovado pela banca avaliadora em: 12/12/2025

BANCA AVALIADORA

Geomar Andre Schreiner - UFFS

Denio Duarte - UFFS

Braulio Adriano de Mello - UFFS

SBC Computing Reviews, 202X, XX:1
ISSN: 2966-3938 • doi: 10.5753/rocs.202X.XXXXXX

Revisão da Literatura

Uma linguagem para a modelagem lógica de Banco de Dados no brModelo Web

Thaila Roanni Schmidt Cavalheiro [Universidade Federal da Fronteira Sul | *thailaroani@gmail.com*]

Instituto ..., Universidade Federal da Fronteira Sul, Endereço, Chapecó, Santa Catarina, CEP, Brasil.

Resumo. A representação de um Banco de Dados pode ocorrer em diferentes níveis: conceitual, lógico e físico, exigindo distintos níveis de abstração para expressar diferentes aspectos do sistema de forma integrada. Considerando a importância da modelagem lógica nesse processo, este trabalho apresenta uma extensão desenvolvida para a ferramenta brModelo Web, que possibilita o desenvolvimento da modelagem lógica de forma descritiva. Seu principal objetivo foi o desenvolvimento de uma linguagem específica para facilitar a criação de tabelas e relacionamentos.
A metodologia adotada envolveu a revisão bibliográfica, a definição da linguagem proposta, a sua implementação no sistema brModelo Web e a avaliação da ferramenta, realizada com estudantes e colaboradores do projeto. Os resultados indicam boa aceitação da abordagem proposta, com destaque para a utilidade prática e a clareza da linguagem, além da preservação da usabilidade original da ferramenta, evidenciando o potencial da solução como complemento ao processo tradicional de modelagem lógica.

Abstract. The representation of a Database can occur at different levels, namely conceptual, logical, and physical, requiring distinct levels of abstraction to express different aspects of the system in an integrated manner. Considering the importance of logical modeling in this process, this work presents an extension developed for the brModelo Web tool, which enables the development of logical modeling in a descriptive way. Its main objective was the development of a domain-specific language to facilitate the creation of tables and relationships.
The adopted methodology involved a bibliographic review, the definition of the proposed language, its implementation in the brModelo Web system, and the evaluation of the tool, carried out with students and project collaborators. The results indicate good acceptance of the proposed approach, highlighting its practical usefulness and the clarity of the language, as well as the preservation of the tool's original usability, demonstrating the potential of the solution as a complement to the traditional logical modeling process.

Palavras-chave: modelagem, lógica, banco, dados, DSL

Keywords: modeling, logic, database, tool, brModelo

Recebido/Received: DD Month YYYY • **Aceito/Accepted:** DD Month YYYY • **Publicado/Published:** DD Month YYYY

1 Introdução

Segundo [Elmasri and Navathe, 2011], os bancos de dados (BD) têm exercido influência significativa no avanço do uso dos computadores. Os autores destacam que essas tecnologias são fundamentais em praticamente todos os setores que fazem uso intensivo de sistemas computacionais, como negócios, comércio eletrônico, engenharia, medicina, genética, direito, educação e biblioteconomia.

A representação de um BD pode ocorrer em diferentes níveis, que vão desde a modelagem conceitual até sua definição física dentro de um sistema gerenciador de banco de dados (SGBD). Nesse processo, ferramentas computacionais assumem um papel importante ao permitir a criação e manipulação dos modelos, seja de forma visual ou textual.

Considerando essa relevância e buscando ampliar as possibilidades de modelagem, este trabalho propõe o desenvolvimento de uma linguagem específica (Domain Specific Language – DSL) integrada ao brModelo Web[1], destinada à definição textual de modelos lógicos de BD.

A motivação para essa proposta surge da necessidade de oferecer aos usuários um meio mais acessível, expressivo e flexível de construir modelos, reduzindo a dependência exclusiva da interface gráfica e permitindo fluxos de trabalho alternativos. A abordagem textual também contribui para maior clareza na especificação e se aproxima de práticas modernas adotadas em linguagens descritivas e ferramentas voltadas ao desenvolvimento de software.

Além disso, a criação de uma DSL integrada ao ambiente do brModelo Web responde a lacunas observadas no uso da ferramenta, como a ausência de um mecanismo estruturado para descrição textual dos modelos, a dificuldade de manipulação em tarefas repetitivas ou de grande escala e a necessidade de um suporte mais adequado para usuários que preferem ou necessitam trabalhar com representações formais.

Com o intuito de avaliar a eficácia da proposta, foram conduzidos experimentos com estudantes e colaboradores envolvidos no desenvolvimento da ferramenta. Os resultados mostraram elevada aceitação quanto à utilidade, clareza e impacto no fluxo de trabalho, indicando que a DSL não apenas amplia as capacidades do brModelo Web, mas também contribui para melhorar a experiência geral do usuário. Os participantes destacaram que a abordagem textual simplifica a criação dos modelos e oferece entendimento mais direto,

[1] `https://www.brmodeloweb.com/lang/pt-br/index.html`

especialmente para usuários com menor familiaridade com interfaces gráficas ou com técnicas avançadas de modelagem.

Por fim, este trabalho está organizado da seguinte forma. Inicialmente, apresenta-se o referencial teórico necessário à compreensão dos conceitos fundamentais relacionados à modelagem de bancos de dados. Em seguida, são discutidos os trabalhos relacionados, situando a proposta no contexto das pesquisas e ferramentas existentes. Posteriormente, descreve-se o processo de implementação e integração da DSL ao brModelo Web, bem como os procedimentos adotados na avaliação experimental. Por fim, são apresentadas as conclusões do trabalho.

2 Referencial Teórico

Nesta seção, serão abordados assuntos necessários para a compreensão deste artigo. Serão apresentados conceitos relacionados à modelagem de BD, com foco nos modelos conceitual e lógico relacional, também na análise da ferramenta brModelo Web e suas funcionalidades.

2.1 Modelagem de Banco de Dados

[Elmasri and Navathe, 2011] citam que o esquema conceitual tem como objetivo um conhecimento completo da estrutura do BD, do significado, dos inter-relacionamentos e das restrições. Ele é uma descrição estável do conteúdo do BD, e pode servir como um veículo de comunicação entre os usuários, os projetistas e os analistas.

Existem diferentes níveis de modelagem, conceitual, lógica e física. Sendo a modelagem Conceitual uma representação mais abstrata, por meio de diagramas como o modelo entidade-relacionamento. A modelagem lógica consiste na transformação dessa estrutura conceitual em um modelo de dados compatível com o tipo de SGBD adotado, sendo o modelo relacional o mais utilizado, no qual os dados são representados por tabelas, chaves e relacionamentos que servirão de base para a implementação. Por fim, a modelagem física transforma o modelo lógico em uma estrutura física específica para o SGBD alvo. No caso de sistemas relacionais, essa etapa define tabelas, colunas, tipos de dados, chaves, índices, restrições e outras configurações que determinam como os dados serão armazenados.

Para o projeto do esquema conceitual, primeiramente identifica-se os componentes básicos do esquema, como tipos de entidades, relacionamentos e atributos. Também é necessário indicar os atributos identificadores das entidades, a cardinalidade e o tipo de participação nos relacionamentos. Além disso, devem ser considerados os tipos de entidades fracas e as hierarquias ou reticulados de especialização/generalização.

A entidade representa um objeto do mundo real com existência própria no BD, como Aluno ou Curso. Os relacionamentos expressam associações entre entidades, como Matrícula, que conecta Aluno e Curso. Os atributos descrevem características das entidades ou relacionamentos, como nome e idade. O atributo identificador distingue unicamente cada instância, por exemplo, o CPF do aluno. A cardinalidade define quantas ocorrências de uma entidade podem associar-se a outra, sendo expressa em formatos como 1:1, 1:N, N:N, 0:1 ou 0:N. Já a participação indica se a presença da entidade no relacionamento é obrigatória (total) ou opcional (parcial).

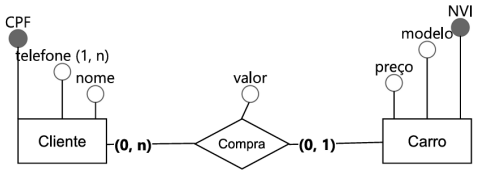

Figura 1. Exemplo Modelo Conceitual

A Figura 1, apresenta um exemplo de modelo conceitual, onde a entidade Cliente (representada por um retângulo) possui os atributos telefone (com cardinalidade 1:N, indicando que um cliente deve ter ao menos um ou vários telefones), nome e CPF como chave primária. A entidade está associada ao relacionamento Compra (representado por um losango), que estabelece uma ligação entre Cliente e Carro, com cardinalidade (0,N) em Cliente, isso significa que um cliente pode realizar zero ou várias compras, e cardinalidade (0,1) em Carro, que significa que cada carro pode estar associado a zero ou apenas uma compra. O relacionamento Compra também possui o atributo valor, que representa o custo da aquisição. Por fim, a entidade Carro é caracterizada pelos atributos preço, modelo e NVI como chave primária.

No nível lógico, adotando o modelo relacional, de acordo com [Ramakrishnan and Gehrke, 2003], esse processo envolve a transformação sistemática de entidades, relacionamentos e restrições em estruturas relacionais que podem ser implementadas em um SGBD.

Cada conjunto de entidades no modelo entidade-relacionamento é convertido em uma relação no modelo lógico relacional, em que os atributos das entidades se tornam colunas e o atributo identificador transforma-se em chave primária. As tuplas da tabela representam as instâncias concretas das entidades. Já os conjuntos de relacionamentos podem ser convertidos em novas relações que incluem as chaves primárias das entidades participantes como chaves estrangeiras, além dos atributos do relacionamento. Essas chaves estrangeiras, em conjunto, constituem uma superchave e, em geral, uma chave candidata, desde que nenhum subconjunto desses atributos seja suficiente para identificar unicamente as instâncias do relacionamento. Quando uma entidade participa múltiplas vezes de um relacionamento com papéis distintos, os nomes dos campos devem refletir esses papéis para evitar ambiguidades.

Quando um relacionamento envolve uma entidade fraca, esta depende de uma entidade forte para ser identificada, o que é representado por uma restrição de chave (seta no diagrama ER). Nesses casos, a chave primária da entidade forte é usada como parte da chave da relação resultante. É possível criar uma tabela exclusiva para o relacionamento ou incorporar seus atributos na tabela da entidade dependente.

Entidades fracas não possuem chave primária própria, dependendo de uma entidade forte para sua identificação. Caso a entidade forte seja deletada, suas entidades fracas associadas também devem ser removidas, garantindo a integridade referencial.

As hierarquias ISA representam heranças entre entidades, nas quais uma subclasse herda os atributos da superclasse. A tradução para o modelo relacional pode gerar tabelas separadas para a superclasse e para cada subclasse, exi-

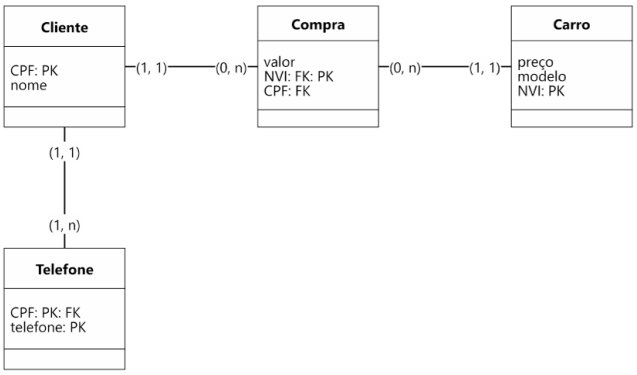

Figura 2. Exemplo Modelo Lógico

gindo junções para consultas completas, ou pode eliminar a tabela da superclasse, incluindo seus atributos diretamente nas subclasses.

Por fim, a agregação, semelhante à entidade associativa, ocorre quando um relacionamento envolve outro relacionamento como se fosse uma entidade, neste caso, cria-se uma tabela com chaves do relacionamento agregado, junto com chaves de entidades externas e atributos descritivos. Se o relacionamento agregado não tem atributos e participa totalmente do novo relacionamento, ele pode ser omitido.

Na Figura 2, apresenta-se a conversão do modelo conceitual para o modelo lógico relacional, onde as entidades e relacionamentos foram transformados em tabelas. A entidade Cliente é representada pela tabela com os atributos nome e chave primária (PK) CPF. O atributo multivalorado telefone foi modelado como uma tabela separada, vinculada à tabela Cliente por meio da chave estrangeira (FK) CPF. A cardinalidade definida no modelo conceitual garante que cada cliente possua ao menos um número de telefone, enquanto a chave primária composta (CPF, telefone) assegura a singularidade dos registros para cada cliente. A tabela Carro mantém os atributos NVI como chave primária, modelo e preço. Já o relacionamento Compra, que possui o atributo valor, foi convertido em uma tabela associativa, conectando as tabelas Cliente e Carro por meio das chaves estrangeiras CPF e NVI. A chave primária NVI na tabela Compra reflete a restrição de que um mesmo carro pode ser adquirido apenas uma vez.

Nesse contexto, o uso de ferramentas computacionais pode auxiliar significativamente na aplicação prática desses conceitos teóricos. Uma dessas ferramentas é o brModelo Web, que será apresentado na próxima seção.

2.2 BrModelo Web

O brModelo Web constitui-se como uma ferramenta computacional brasileira de código aberto voltada à modelagem conceitual de dados, com ênfase na elaboração de diagramas Entidade-Relacionamento (ER). Trata-se de uma reinterpretação em ambiente web do tradicional brModelo desktop, amplamente utilizado no contexto acadêmico brasileiro para o ensino de modelagem de BD. A nova versão, acessível por meio do repositório oficial no GitHub[2] foi desenvolvida para ampliar a acessibilidade e usabilidade do brModelo em ambiente web.

A plataforma permite a construção visual de modelos

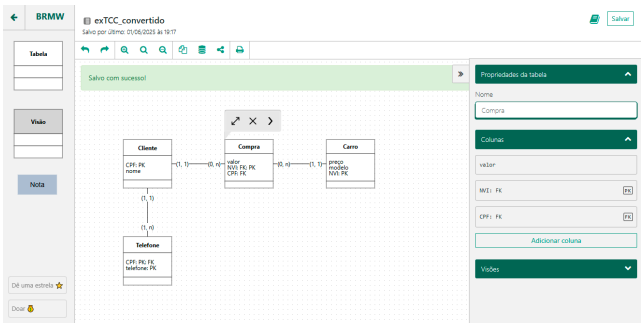

Figura 3. Interface do brModelo Web

conceituais com suporte a entidades, relacionamentos, atributos, especializações e generalizações, além de possibilitar a geração automática dos modelos lógico relacional e físico correspondentes, também sendo possível a criação dos mesmos. Os modelos podem ser exportados em linguagem SQL, o que facilita sua integração com sistemas gerenciadores de bancos de dados. Na Figura 3, apresenta-se um exemplo de sua interface com a construção do modelo lógico apresentado anteriormente.

Sua implementação baseia-se em tecnologias como Angular[3], framework para desenvolvimento de interfaces web dinâmicas e responsivas. A ferramenta também faz uso da biblioteca JointJS[4] , responsável pela construção e manipulação dos diagramas gráficos, permitindo a criação, edição e visualização dos elementos do modelo de dados de forma interativa. Além disso, sua arquitetura modular é organizada em componentes e serviços independentes, o que facilita a evolução do sistema. Novas funcionalidades podem ser adicionadas por meio da criação de módulos integrados à interface e ao núcleo da aplicação. Entre essas possibilidades, este trabalho propõe o desenvolvimento de uma DSL voltada à definição textual de modelos lógicos relacionais de BD, cuja integração ao brModelo Web seria o foco central.

3 Trabalhos Relacionados

Nesta seção, serão apresentados e discutidos trabalhos relacionados que apresentam semelhanças com o projeto desenvolvido, especialmente no que diz respeito às características e funcionalidades da ferramenta proposta. Para a identificação desses trabalhos, foram realizadas buscas no Google Scholar utilizando termos relacionados à modelagem BD e linguagens textuais de definição de esquemas. A partir dessa pesquisa, foram identificados diversos artigos, dentre os quais se destacam, por sua relevância, o trabalho TXL: A Language for Programming Language Tools and Applications [Cordy, 2006], a ferramenta de modelagem de bancos de dados relacionais brModelo v3 [Candido and Mello, 2013] e o estudo Textual Approach for Designing Database Conceptual Models: A Focus Group [Lopes *et al.*, 2021]. Este último foi selecionado para análise mais detalhada devido à sua abordagem textual voltada para a modelagem de dados e por apresentar conceitos aplicáveis ao sistema proposto.

O Dbdiagram[5] é uma ferramenta de código aberto gratuita para desenhar diagramas Entidade-Relacionamento (ERD) de BD por meio da escrita de código em sua própria

[2] https://github.com/brmodeloweb/brmodelo-app

[3] https://angular.dev

[4] https://www.jointjs.com

[5] https://dbdiagram.io/

DSL. Seu principal diferencial está na oferta de um plano premium que disponibiliza funcionalidades mais avançadas.

A DSL utilizada pela ferramenta é inspirada na sintaxe de linguagens de definição de esquemas relacionais, como o SQL, permitindo a definição de tabelas e relacionamentos de forma textual e concisa.

Já o [Čeliković *et al.*, 2014] apresentam uma ferramenta chamada System Modeling Tool (MIST). O MIST foi desenvolvido para modelar sistemas de inovação dentro de empresas e organizações. Ele utiliza uma DSL chamada EERDSL, baseada no modelo Extended Entity-Relationship (EER) aprimorado. Entretanto, a ferramenta propõe uma abordagem voltada para o modelo conceitual de dados, enquanto o projeto desenvolvido tem como foco a modelagem em nível lógico.

E, por fim, o ERtext [Lopes *et al.*, 2021] é uma ferramenta desenvolvida com o objetivo de apoiar o ensino e a prática da modelagem conceitual de BDs relacionais por meio de uma abordagem textual. A ferramenta é baseada em uma linguagem DSL, permitindo a criação de esquemas Entidade-Relacionamento. Diferente de outras soluções voltadas exclusivamente para ambientes profissionais ou acadêmicos mais avançados, o ERtext propõe uma interface que unifica o estilo textual das etapas de modelagem conceitual, lógica e física, promovendo uma transição entre essas fases.

A linguagem utilizada pela ferramenta foi concebida com base na infraestrutura do Xtext, possibilitando a definição de sua gramática de forma declarativa. A instrução *grammar* define o nome da linguagem, enquanto a cláusula *with* estabelece herança da gramática padrão *Terminals*, fornecendo regras léxicas básicas como identificadores e literais. A geração da Árvore de Sintaxe Abstrata (AST) é realizada pela instrução *generate*, que estrutura os componentes internos da linguagem.

No nível estrutural, a regra de entrada da DSL define a composição dos arquivos, que devem conter, obrigatoriamente, um objeto *Domain*, seguido de zero ou mais objetos *Entity* e *Relation*. As palavras reservadas são indicadas por aspas, como *"Entities {"*, que sinaliza o início de um bloco de definição de entidades. A multiplicidade dos elementos é indicada por operadores específicos: $*$ (zero ou muitos), $+$ (um ou muitos), e $?$ (zero ou um).

A Figura 4 apresenta a definição da gramática da DSL, evidenciando a composição dos arquivos, a hierarquia entre *Domain*, *Entity* e *Relation*, e a sintaxe utilizada para atributos, multiplicidades e tipos de dados.

Quanto à atribuição, o símbolo $=$ indica a exigência de um único valor, enquanto $+=$ permite múltiplos registros. A definição de uma entidade é feita por meio da palavra-chave *Entity*, seguida por um identificador, com herança opcional utilizando a palavra *is*. Dentro do bloco da entidade, delimitado por chaves, são listados os atributos, sendo obrigatório ao menos um. Embora não seja obrigatório conter um identificador (em virtude da existência de entidades fracas), é possível marcar atributos com o modificador *isIdentifier* para indicar chaves primárias.

Relacionamentos são definidos no bloco *Relationships*, com uma declaração opcional de nome. Os lados do relacionamento são representados por instâncias da regra *RelationSide*, que especifica a cardinalidade como *(0:1), (1:N)*)

```
grammar org.xtext.university.ertext.ERtext
with org.eclipse.xtext.common.Terminals
generate ERtext "http://www.xtext.org/unipampa/ertext/
    ERtext"
ERModel:
    domain=Domain ';'
    ('Entities' '{') entities+=Entity+ ('}' ';')
    ('Relationships' '{') relations+=Relation*
      ('}' ';');
Domain:
    'Domain' name=ID;
Attribute:
    name=ID type=DataType (isKey?='isIdentifier')?;
Entity:
    name=ID ('is' is+=[Entity])*
    ('{' attributes+=Attribute
    (',' attributes+=Attribute)*'}')?;
Relation:
    (name=ID)? ('[' leftEnding=RelationSide
    'relates'
    rightEnding=RelationSide ']')
    ('{' attributes+=Attribute
    (',' attributes+=Attribute)* '}')*;
RelationSide:
    cardinality=('(0:1)'|'(1:1)'|'(0:N)'|'(1:N)')
    target=[Entity] | target=[Relation];
enum DataType:
    INT='int'          | DOUBLE='double'      |
    MONEY='money'      | STRING='string'      |
    BOOLEAN='boolean'  | DATETIME='datetime'  |
    BLOB='file';
```

Figura 4. Exemplo DSL ERtext

e o alvo *(target)*, que pode ser uma *Entity* ou até mesmo outra *Relation*, possibilitando a modelagem de relacionamentos ternários. A linguagem também conta com um conjunto enumerado de tipos de dados *(DataType)*, cuja sintaxe utiliza o operador $|$ para separação de alternativas.

Assim, a abordagem do ERtext utiliza uma linguagem formal para modelagem conceitual de BD, com foco na clareza e na validação estrutural, sendo destinada principalmente a usos didáticos e experimentais.

Vale destacar que, embora existam diferentes estratégias para o mapeamento de hierarquias entre entidades, como a utilização de uma única tabela unificada para representar toda a hierarquia (tabela mesclada), a ferramenta adota exclusivamente a abordagem de gerar uma tabela separada para cada entidade. Dessa forma, não são tratadas outras alternativas de transformação, como a fusão de tabelas ou representações hierárquicas mais complexas, limitando a flexibilidade do modelo lógico gerado quanto a essas variações estruturais.

A partir da Tabela 1, observa-se que as ferramentas analisadas apresentam propostas e públicos-alvo distintos. O Dbdiagram.io destaca-se pelo foco na modelagem lógica e na geração automática de código SQL, sendo mais voltado a desenvolvedores e analistas que buscam rapidez na criação de esquemas. A MIST, por sua vez, adota uma abordagem conceitual baseada em Extended Entity-Relationship, direcionando-se a organizações que lidam com sistemas mais complexos e demandam maior expressividade semântica. Já o ERtext apresenta uma proposta voltada ao ensino e à prática acadêmica, oferecendo suporte à modelagem conceitual com posterior transformação para o nível lógico.

De forma geral, nota-se que, embora todas as ferramentas utilizem abordagens textuais, elas se diferenciam quanto ao nível de abstração priorizado e aos objetivos de uso. Essa análise evidencia a lacuna existente para uma solução integrada ao brModelo Web que permita a definição textual da modelagem lógica relacional, mantendo alinhamento com uma ferramenta amplamente utilizada no contexto acadêmico.

Tabela 1. Comparação entre ferramentas de modelagem de dados

Aspecto	Dbdiagram.io	MIST	ERtext
Público-alvo	Desenvolvedores e analistas de dados	Empresas e organizações que modelam sistemas complexos	Ensino e prática acadêmica
Nível de abstração	Modelagem lógica com geração automática de SQL	Modelagem conceitual baseada em Extended ER	Modelagem conceitual e transformação para lógica relacional
Diferenciais	Plano premium, geração automática de código SQL	DSL EERDSL para inovação em sistemas	Linguagem baseada em Xtext
Foco do sistema	Geração de diagramas e código	Modelagem de sistemas de inovação	Ensino e aprendizado da modelagem conceitual

4 LogiText

Esta seção apresenta as fases de desenvolvimento do projeto, descrevendo a definição da DSL, sua implementação ao sistema e a avaliação dos resultados obtidos.

4.1 Linguagem

A nova DSL, denominada *LogiText*, permite a definição de tabelas, chaves e relacionamentos de acordo com as regras da modelagem lógica relacional.

Trata-se de uma linguagem objetiva, com sintaxe simples e próxima da linguagem natural. Tendo como referência Mernik *et al*. [2005], que destaca a importância de notações específicas, pois estão diretamente relacionadas ao aumento de produtividade associado ao uso de DSLs.

A linguagem foi projetada para permitir o desenvolvimento por usuários com pouca experiência em programação, ou até mesmo por usuários finais com conhecimento limitado do domínio técnico.

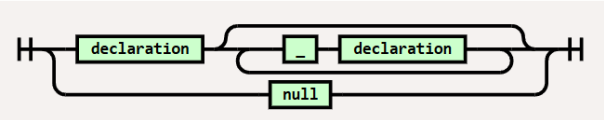

Figura 5. *Main*

A Figura 5 apresenta o diagrama de sintaxe gerado pelo *toolkit* Nearley, que descreve visualmente as regras da gramática da DSL *LogiText*. Ela mostra que a estrutura do *main* consiste em uma declaração, seguida de zero ou mais pares, ou seja, isso significa que um arquivo LogiText pode conter diversas declarações ou nenhuma.

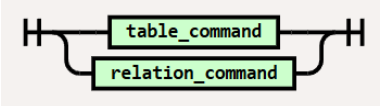

Figura 6. *Declaration*

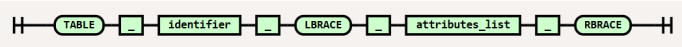

Figura 7. *Table Command*

A Figura 6 indica que uma declaração pode ser um comando de tabela ou um comando de relação. Cada tabela é definida pelo comando *Table*, seguido pelo seu nome identificador. Após a definição do nome, abre-se um bloco entre chaves {}, onde são especificados os atributos da tabela, como apresentado na Figura 7. A DSL é *case sensitive*, ou seja, o comando *Table* deve ser descrito com letra maiúscula.

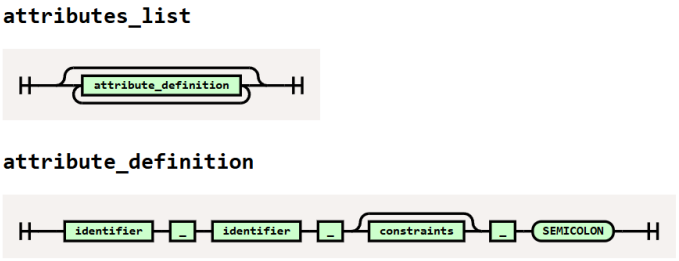

Figura 8. *Attributes*

A Figura 8 apresenta a regra de definição dos atributos. Uma tabela pode conter zero ou mais atributos, e cada atributo é descrito pelo seu nome, seguido do tipo e, opcionalmente, de restrições separadas por vírgulas dentro de colchetes, finalizando com ponto e vírgula. O tipo do atributo pode ser *int*, *float*, *varchar*, *char*, *boolean* ou *date*.

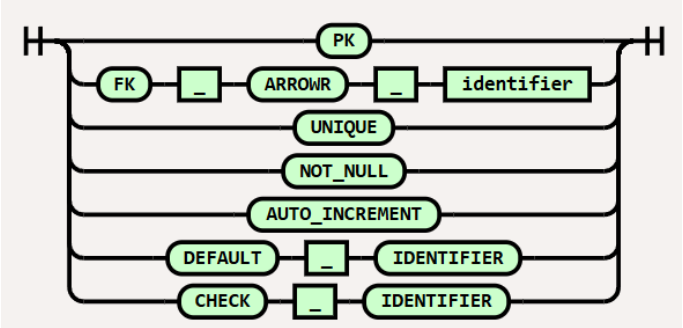

Figura 9. *Constraints*

A Figura 9 mostra que as restrições possíveis: [*pk*] para chave primária, [*fk -> tabela_de_origem*] para chave estran-

geira, onde a seta indica a tabela referenciada, [*unique*] para valor único, [*not null*] para atributo obrigatório, [*auto increment*] para valor gerado automaticamente, [*default*] para definir valor padrão e [*check*] para impor uma condição lógica. Contudo, a restrição *check* ainda não está disponível e será implementada futuramente.

A Figura 10 apresenta a definição dos relacionamentos pelo comando *Relation*, o formato utilizado é ”Relation tabela_1 (pk_tabela_1) (cardinalidades) tabela_2(fk_tabela_2)”. Nesse formato, a chave primária da primeira tabela (tabela_1) é relacionada à chave estrangeira da segunda tabela (tabela_2).

A Figura 11 apresenta a notação de cardinalidades utilizada pela DSL. Nessa notação, o símbolo “<” indica que o lado esquerdo do relacionamento corresponde a “N” (muitos), enquanto o símbolo “>” indica que o lado direito representa “N”. Os números 0 e 1, por sua vez, indicam a cardinalidade mínima do lado que não é N. Assim, expressões como “<0” e “<1” significam, respectivamente, 0:N e 1:N, enquanto “0>” e “1>” representam 0:N e 1:N no lado direito. Com essa combinação de símbolos, cada padrão expressa a cardinalidade completa de um relacionamento.

Por exemplo, em “Relation Compra(cpf_cliente) <0-1 Cliente(cpf);”, a notação “<0-1” indica que o lado da tabela Compra corresponde ao lado N, ao passo que o número 1 indica que cada Cliente participa com, no mínimo, uma instância do relacionamento. Isso significa que um Cliente pode estar associado a várias ou nenhuma Compra, enquanto cada Compra está vinculada exatamente a um único Cliente.

Na Figura 12 é apresentado um exemplo da linguagem proposta, em que são definidas as tabelas *Cliente, Compra, Carro* e *Telefone* juntamente com seus respectivos atributos. Cada atributo é seguido do seu tipo e, opcionalmente, de características adicionais entre colchetes. Após a definição das tabelas, são descritos os relacionamentos entre elas, especificando as cardinalidades e os atributos envolvidos em cada ligação.

A Figura 13 apresenta o resultado da aplicação da linguagem de exemplo no ambiente do brModelo Web, evidenciando o modelo lógico gerado a partir da especificação textual. Observa-se que as tabelas e relacionamentos definidos na DSL são corretamente refletidos na representação gráfica, mantendo coerência entre a descrição textual e o diagrama visual. Essa integração permite que o usuário utilize a linguagem para construir o modelo de forma textual e, simultaneamente, acompanhe sua visualização gráfica, facilitando a validação da estrutura definida e reforçando a consistência entre as duas abordagens de modelagem.

4.2 Implementação

4.2.1 Editor

Na implementação do BrModelo Web, o CodeMirror[6] é utilizado como editor de código para a DSL que representa os modelos lógicos. Ele fornece destaque de sintaxe, fechamento automático de colchetes e outras funcionalidades típicas de editores de código, permitindo que o usuário visualize e edite o modelo de forma textual.

A integração com o Angular é feita através de uma diretiva personalizada, que cria o editor a partir de um *<textarea>* e sincroniza suas alterações com o *LogicService*. Dessa forma, qualquer modificação feita no diagrama visual é refletida automaticamente no código e vice-versa, garantindo consistência entre a representação gráfica e a textual do modelo lógico.

4.2.2 Parser

A implementação da DSL LogiText inicia-se pela construção do parser responsável por interpretar os comandos da linguagem. Para isso, foi utilizado o Nearley[7], um *parser generator* baseado em gramáticas do tipo Earley. Ele permite definir a sintaxe da linguagem por meio de regras declarativas escritas em um arquivo *.ne*, a partir do qual o próprio Nearley gera automaticamente o código do parser em JavaScript. Essa abordagem foi escolhida porque o Nearley oferece suporte a gramáticas ambíguas, é de fácil integração com o ecossistema Node.js e possui uma sintaxe expressiva que facilita a definição de linguagens personalizadas.

Após a geração do parser pelo Nearley, a entrada analisada é convertida em uma árvore sintática abstrata (AST). Na LogiText, essa AST é definida pelas regras do arquivo *grammar.ne*, que inclui a gramática da linguagem e seus *postprocessors*. Estes transformam sequências de *tokens* em objetos estruturados, representando tabelas, atributos, restrições e relações. O Nearley compila a gramática em grammar.js, que contém a versão JavaScript da gramática.

4.2.3 Interpretação Semântica

Uma vez construída, a AST é entregue ao módulo responsável pela análise semântica da linguagem: a classe *SemanticInterpreter*. Essa etapa é fundamental porque a gramática garante apenas que a entrada é sintaticamente válida, entretanto, erros como “coluna duplicada”, “tabela referenciada inexistente” ou “tipo inválido” só podem ser identificados a partir de uma inspeção semântica. O interpretador percorre cada nó da AST e aplica regras de consistência, garantindo que o modelo lógico gerado seja coerente dentro da própria DSL LogiText.

O método *execute()* é o ponto de entrada dessa etapa. Ele recebe a AST produzida pelo Nearley e, para cada nó, verifica o tipo do comando interpretado anteriormente pelo parser. Quando encontra nós do tipo *table*, transfere a validação para *_checkTable()*, que analisa nomes duplicados, tipos de dados, restrições (pk, fk, default, etc.) e converte cada atributo em um objeto consistente para uso interno.

De forma complementar, o método *_checkRelation()* é responsável pela validação dos relacionamentos entre tabelas. Como relações dependem de tabelas pré-definidas na AST, essa função assegura que ambas as tabelas existam, que as colunas referenciadas realmente estão presentes e que a cardinalidade declarada corresponda a um dos padrões aceitos pela linguagem. Caso contrário, uma exceção semântica é lançada.

Ao final, tanto tabelas quanto relacionamentos são agregados nas listas *this.tables* e *this.relations*, que representam o resultado semântico final, uma versão pronta para ser utilizada pela etapa seguinte da LogiText.

[6]https://codemirror.net

[7]`https://nearley.js.org`

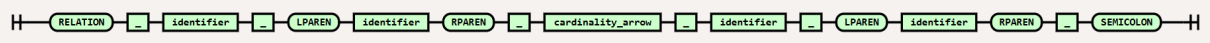

Figura 10. *Relations*

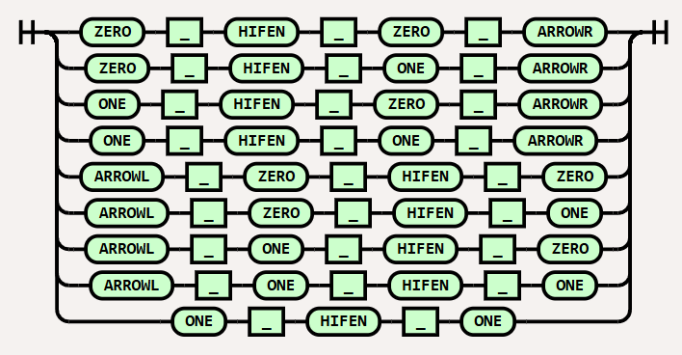

Figura 11. *Cardinality*

4.2.4 Geração de Diagramas

O BrModelo Web utiliza a JointJS, uma biblioteca JavaScript para criação de diagramas, na construção do modelo lógico gerado pelo *SemanticInterpreter*. Nesse contexto, a classe *DiagramGeneratorLogical* converte os objetos semânticos da AST em elementos visuais que representam tabelas e relacionamentos no diagrama.

O construtor da classe recebe como parâmetro o *logicService* (ls), que funciona como o cérebro da parte lógica do BrModelo Web. Ele mantém o estado do modelo lógico, controla seleção de elementos, criação e edição de tabelas e relacionamentos, aplica eventos de interface e sincroniza a representação visual com a DSL LogiText. Em resumo, ele conecta a interface gráfica com a lógica semântica do modelo.

O método principal, *generate*(), é responsável por receber a AST, processando suas tabelas e relações. Ele percorre os nós da AST, chamando *createTable()* para cada tabela e *createRelation()* para cada relacionamento, consolidando a estrutura visual completa. Ao final, retorna objetos representando as tabelas e relações, permitindo que outros módulos da aplicação possam acessar ou manipular esses elementos de forma estruturada.

O método *createTable()* desempenha a função de traduzir cada objeto da tabela em um elemento visual do JointJS. Para posicionamento, ele utiliza uma lógica simples de grade, dispondo até três tabelas por linha e calculando a posição vertical conforme novas linhas são necessárias. Cada coluna da tabela é processada individualmente, suas propriedades semânticas são mapeadas para atributos do objeto. O método também converte colunas em strings formatadas para exibição, destacando chaves primárias e estrangeiras.

Caso a tabela já exista no grafo, ele atualiza suas propriedades e atributos, em vez de criar um novo elemento, disparando eventos internos do JointJS para que a interface responda às mudanças. Caso contrário, um novo elemento é criado, com dimensões ajustadas ao número de atributos, e seus objetos internos são configurados para permitir edição posterior pelo usuário. As chaves estrangeiras são vinculadas às tabelas referenciadas, associando o *fkId* ao *ID* do elemento correspondente no grafo, estabelecendo assim a relação visual necessária.

O método *createRelation()* é responsável por traduzir

Text Editor | Elements

```
1  Table Cliente {
2    cpf int [pk];
3    nome varchar;
4  }
5
6  Table Carro {
7    preco float;
8    modelo varchar;
9    id int [pk];
10 }
11
12 Table Telefone {
13   cpf int [pk, fk -> Cliente];
14   telefone int [pk];
15 }
16
17 Table Compra {
18   valor float;
19   id_carro int [pk, fk ->
   Carro];
20   cpf_cliente int [fk ->
   Cliente];
21 }
22
23 Relation Telefone(cpf) <1-1
   Cliente(cpf);
24
25 Relation Compra(id_carro) <0-1
   Carro(id);
26
27 Relation Compra(cpf_cliente)
   <0-1 Cliente(cpf);
```

Figura 12. Exemplo de linguagem

os relacionamentos semânticos em linhas de ligação no diagrama. Para cada relação, ele identifica as tabelas de origem e destino no grafo e calcula os rótulos de cardinalidade. Os elementos visuais de ligação são então criados mostrando a cardinalidade próxima às extremidades da relação.

Dessa forma, a classe *DiagramGeneratorLogical* assegura que cada tabela e relação seja traduzida de maneira precisa para o diagrama visual, mantendo todas as propriedades semânticas originais e permitindo interatividade na edição do modelo lógico dentro do BrModelo Web.

4.2.5 Função Reversa

O *reverseToDSL* é responsável por gerar a representação textual da DSL a partir do modelo visual do JointJS. Ele percorre todas as tabelas e colunas do diagrama, identifica os tipos e restrições e converte cada coluna em uma linha de código da DSL. Quando uma coluna possui uma FK com referência a outra tabela, a função cria automaticamente um relacionamento correspondente, inferindo uma cardinalidade padrão,

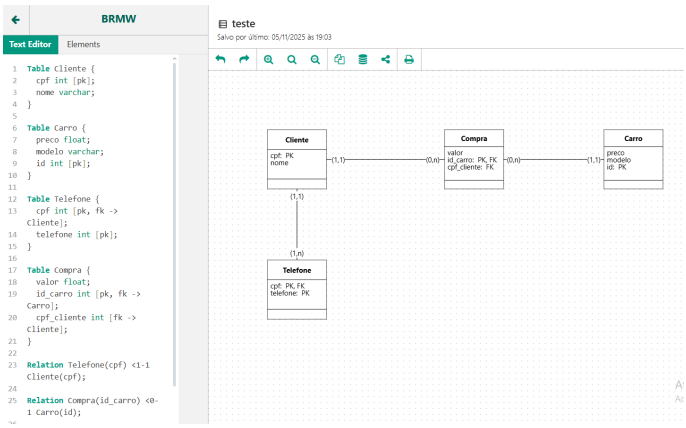

Figura 13. Exemplo de linguagem

como o brModelo Web já fazia, garantindo que a relação entre tabelas seja representada corretamente no código.

No final, toda a AST gerada é transformada em texto e atualiza o editor de código (CodeMirror), mantendo sincronizadas a versão visual e a textual do modelo lógico. Dessa forma, qualquer alteração feita no diagrama, seja adicionando colunas ou ajustando FKs, é refletida imediatamente na DSL.

5 Avaliação da solução

A avaliação da linguagem desenvolvida e de sua integração ao BrModelo Web foi realizada com a participação de estudantes da Universidade e colaboradores vinculados ao projeto, totalizando 10 respostas. O objetivo consistiu em verificar a utilidade prática da abordagem descritiva proposta, bem como a clareza da DSL e a adequação de sua implementação na ferramenta.

Para condução da análise, os participantes utilizaram a extensão textual implementada, realizaram testes livres e, posteriormente, responderam a um formulário estruturado contendo questões objetivas e espaços para comentários qualitativos. As perguntas foram organizadas de forma a contemplar quatro dimensões principais: utilidade da proposta, clareza e objetividade da DSL, funcionamento da implementação e impacto na usabilidade do BrModelo Web.

As respostas foram coletadas por meio de escalas de avaliação do tipo *Likert*, adaptadas para cada dimensão analisada. Para a utilidade da proposta, as opções variaram entre Muito útil, Útil, Adequada, mas pode ser aprimorada, Pouco útil e Não adequada. Na dimensão de clareza e objetividade da DSL, os participantes escolheram entre Concordo totalmente, Concordo, Neutro, Discordo e Discordo totalmente. Quanto ao funcionamento da implementação e ao impacto na usabilidade do BrModelo Web, as respostas foram classificadas como Sim, Parcialmente ou Não. Além disso, cada questão incluía um espaço aberto para comentários qualitativos, permitindo que os participantes detalhassem suas impressões, sugestões e eventuais dificuldades encontradas durante a utilização da ferramenta.

Na Figura 14 é possível observar os resultados obtidos a partir da avaliação realizada com os participantes, considerando as diferentes dimensões analisadas.

Com base nas respostas obtidas para a dimensão referente à utilidade da nova abordagem descritiva, observou-se que 90% dos participantes consideraram a proposta muito

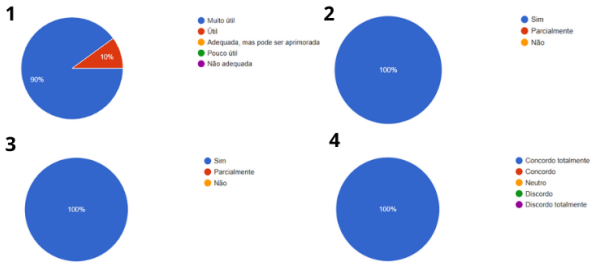

Figura 14. Resultados da avaliação da solução

útil, enquanto 10% a avaliaram como útil. Esses resultados indicam que a solução apresenta potencial para facilitar a definição de modelos lógicos e ser incorporada de forma positiva ao fluxo de trabalho dos usuários. Além disso, os participantes destacaram que a abordagem "facilita a criação e entendimento do que está acontecendo por quem não é tão familiarizado com o tema ou programação", reforçando sua acessibilidade e aplicabilidade. O gráfico 1 apresenta a distribuição completa das respostas para essa questão.

Em relação à clareza e objetividade da DSL, 100% dos participantes relataram uma experiência positiva. Esse resultado sugere que a DSL é compreensível e acessível. O gráfico 2 sintetiza as respostas deste aspecto.

Em relação ao funcionamento da extensão textual integrada ao BrModelo Web, os participantes relataram que, embora a implementação seja funcional e permita a construção dos modelos lógicos conforme proposto, ainda foram identificadas algumas inconsistências durante o uso. Entre os aspectos mencionados, destacam-se principalmente dois pontos.

O primeiro refere-se a problemas de sincronia entre o editor visual e o editor textual. Em determinadas situações, alterações realizadas no diagrama, como a adição de novos atributos, não eram imediatamente refletidas no *Text Editor*, sendo atualizadas apenas após alguma interação gráfica adicional, como o movimento manual de um elemento. Isso ocorre porque a função *reverseToDSL* é acionada apenas quando há interação na interface. Como os atributos são adicionados pelo *sidebar*, essa atualização automática não acontece, resultando na falta de sincronia entre as duas representações do modelo.

O segundo ponto diz respeito ao tratamento de tipos no editor de colunas. Durante a edição por meio do menu de propriedades, o tipo varchar é representado no formato "varchar(n)". No entanto, o caractere "(" é considerado inválido pela DSL, o que faz com que edições legítimas realizadas na interface visual resultem em códigos incompatíveis com a sintaxe textual, ocasionando a quebra temporária da linguagem.

Essas observações indicam que a extensão está operacional, mas ainda demanda ajustes para garantir maior robustez. O gráfico 3 apresenta a distribuição das respostas referentes a funcionalidade da ferramenta.

A análise sobre o impacto da integração da DSL na usabilidade geral do BrModelo Web revelou que 100% dos participantes consideraram que a usabilidade original foi preservada, como apresentado no gráfico 4.

Além disso, os comentários qualitativos reforçam essa percepção, indicando que a integração "melhorou muito a experiência do usuário, e ainda criou a possibilidade de de-

senvolver várias outras funcionalidades". Os participantes também destacaram que "A linguagem é muito útil, e facilita o uso da ferramenta. A descrição textual é muito mais simples que manipular os elementos gráficos", evidenciando ganhos de praticidade e eficiência no processo de modelagem.

Houve também uma sugestão de melhoria relacionada ao aprimoramento das funcionalidades da ferramenta. O participante propôs a integração de uma IA capaz de auxiliar na construção dos diagramas por meio de descrições via *prompt*, tornando o processo mais intuitivo e automatizado. Além disso, sugeriu que a disponibilização de exemplos de uso e orientações operacionais poderia facilitar o entendimento e a usabilidade, especialmente para usuários iniciantes.

A avaliação demonstrou que a integração da DSL ao BrModelo Web foi positiva e bem aceita pelos usuários, reforçando seu potencial para aprimorar o processo de modelagem lógica. Embora tenham sido identificados pontos que requerem ajustes, esses aspectos não comprometem a proposta e indicam caminhos para refinamento da implementação. A sugestão de incluir recursos adicionais, como apoio por IA e exemplos guiados, aponta oportunidades futuras para fortalecer ainda mais a experiência de uso e ampliar as possibilidades da ferramenta.

6 Conclusão

Este trabalho apresentou o desenvolvimento de uma DSL destinada à modelagem lógica de BD, integrada ao ambiente brModelo Web. A proposta teve como motivação a oferta de um mecanismo textual simples, estruturado e expressivo que complementasse o modelo visual tradicional da ferramenta, ampliando suas possibilidades de uso e aproximando-a de abordagens modernas baseadas em DSLs.

Foram discutidas as principais decisões de projeto da linguagem, a definição da gramática utilizando Nearley e as etapas de análise responsáveis por validar a estrutura do código e a consistência dos relacionamentos definidos pelo usuário. A implementação do *SemanticInterpreter* permitiu organizar a AST resultante em objetos semânticos coesos, assegurando a correspondência entre a descrição textual e a representação lógica interna.

A integração com o brModelo Web foi realizada por meio dos módulos de geração de diagramas e da função reversa. A classe DiagramGeneratorLogical mostrou-se essencial para traduzir objetos semânticos em elementos visuais utilizando JointJS, preservando propriedades como chaves primárias, estrangeiras, cardinalidades e demais atributos presentes no modelo. De forma complementar, o módulo reverseToDSL garantiu a sincronização entre as representações visual e textual, permitindo ao usuário alternar e editar o modelo em qualquer uma das interfaces sem perda de informação.

A partir da integração bidirecional entre código e diagrama, o BrModelo Web passa a oferecer um fluxo de modelagem lógica mais completo, acessível e coerente, atendendo tanto usuários que preferem interações visuais quanto aqueles que se beneficiam de uma especificação textual. Os experimentos demonstram que a integração está funcional, refletindo corretamente alterações em qualquer uma das representações.

Implementações futuras poderão corrigir observações identificadas durante a avaliação, como questões de sincronia e tratamento de tipos, além de adicionar funcionalidades, como a restrição *check*, ampliando a robustez e a expressividade da DSL LogiText.

Referências

Candido, C. H. and Mello, R. d. S. (2013). Ferramenta de modelagem de bancos de dados relacionais brmodelo v3. In *Anais do Simpósio Brasileiro de Bancos de Dados*, Brasil.

Čeliković, M., Dimitrieski, V., Aleksić, S., Ristić, S., and Luković, I. (2014). A dsl for eer data model specification.

Cordy, J. R. (2006). Txl: A language for programming language tools and applications. *Science of Computer Programming*, 61(3):190–210.

Elmasri, R. and Navathe, S. B. (2011). *Sistemas de banco de dados*. Pearson Prentice Hall, São Paulo, 6 edition. Revisão técnica de Enzo Seraphim e Thatyana de Faria Piola Seraphim.

Lopes, J., Bernardino, M., Basso, F., and Rodrigues, E. (2021). Textual approach for designing database conceptual models: A focus group. In *MODELSWARD*, pages 171–178.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4).

Ramakrishnan, R. and Gehrke, J. (2003). *Database Management Systems*. McGraw-Hill, New York, 3 edition.