UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

UMA ABORDAGEM PARA AVALIAR A EXPRESSIVIDADE DE GRAMÁTICAS DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO: UM ESTUDO DE CASO COM SQL

GÉSSICA MONIQUE DA SILVA ALVES

FRANKLIN RAMALHO
ORIENTADOR

CAMPINA GRANDE
DEZEMBRO - 2022

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem para Avaliar a Expressividade de Gramáticas de Linguagens Específicas de Domínio: Um Estudo de Caso com SQL

Géssica Monique da Silva Alves

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Franklin Ramalho
(Orientador)

Campina Grande, Paraíba, Brasil

A474a Alves, Géssica Monique da Silva.
Uma abordagem para avaliar a expressividade de gramáticas de linguagens específicas de domínio: um estudo de caso com SQL / Géssica Monique da Silva Alves. – Campina Grande, 2023.
92 f.: il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2023.
"Orientação: Prof. Dr. Franklin de Souza Ramalho"
Referências.

1. Linguagens de Programação de Computador. 2. Expressividade da Linguagem. 3. Avaliação de *Domain-Specific Language*. 4. Métricas Baseadas em Gramáticas. I. Ramalho, Franklin de Souza. II. Título.

CDU 004.43(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA MARIA ANTONIA DE SOUSA CRB 15/398

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
POS-GRADUACAO CIENCIAS DA COMPUTACAO
Rua Aprigio Veloso, 882, - Bairro Universitario, Campina Grande/PB, CEP 58429-900

FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

GÉSSICA MONIQUE DA SILVA ALVES

UMA ABORDAGEM PARA AVALIAR A EXPRESSIVIDADE DE GRAMÁTICAS DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO: UM ESTUDO DE CASO COM SQL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 22/12/2022

Prof. Dr. FRANKLIN DE SOUZA RAMALHO, Orientador, UFCG

Prof. Dr. TIAGO LIMA MASSONI, Examinador Interno, UFCG

Prof. Dr. PAULO EDUARDO E SILVA BARBOSA, Examinador Externo, UEPB

Documento assinado eletronicamente por **TIAGO LIMA MASSONI**, **COORDENADOR(A) ADMINISTRATIVO(A)**, em 24/12/2022, às 10:42, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **FRANKLIN DE SOUZA RAMALHO**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 26/12/2022, às 17:04, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Resumo

Domain-Specific Languages (DSLs) são definidas como linguagens projetadas com foco em um domínio de problema específico. Atualmente, existe uma grande variedade de DSLs, como HTML para páginas web, XML para codificação de dados, SQL para consultas de bancos de dados, Latex, Mathematica, Verilog, entre outras. DSLs são capazes de contribuir em ganhos substanciais relacionados à abstração, compreensão e criação de simples notações no desenvolvimento de artefatos de *software*. Além disso, no trabalho de desenvolvedores, DSLs estão sendo amplamente utilizadas na resolução de problemas, em virtude de possuírem conceitos e termos que lhes são familiares. No entanto, a avaliação de DSLs sob características que afetam a capacidade de escrita em uma linguagem, como por exemplo, sua expressividade, ainda é uma área insuficientemente investigada e pouco apresentada em trabalhos. Por outro lado, quando essas avaliações são propostas, são utilizadas abordagens direcionadas especificamente para as linguagens em estudo, ou seja, não mostram uma análise baseada em métricas que possam validar qualquer tipo de DSL. Considerando esse problema, o presente estudo teve como objetivo desenvolver uma abordagem mediante um estudo de caso com SQL para avaliações de DSLs baseada em suas gramáticas, através da propositura e a aplicação de métricas quantitativas relacionadas à característica de expressividade. Os resultados do estudo apontam que o ganho de expressividade foi observada especialmente pelo aumento significativo 160 novos terminais na versão de SQL de 1999. Estes novos terminais estão relacionados à categorias como tipos de dados, comandos, expressões e exceções que foram adicionados à linguagem. Este trabalho pode ajudar pessoas interessadas na área de DSLs e linguagens em geral, para avaliação de qualidade de expressividade da sua linguagem em estudo, sob uma nova visão que toma como base a análise das gramáticas que são desenvolvidas.

Palavras-Chave: expressividade da linguagem; avaliação de *Domain-Specific Language*; métricas baseadas em gramáticas.

Abstract

Domain-Specific Languages (DSLs) are defined as designed languages focused on a specific problem domain. Currently, there is a wide variety of DSLs, such as HTML for web pages, XML for data encoding, SQL for database queries, Latex, Mathematica, and Verilog, among others. DSLs can contribute to substantial gains related to the abstraction, understanding, and creation of simple notations in the development of software artifacts. Furthermore, in developers' work, DSLs are widely used in problem-solving, because they have concepts and terms familiar to them. However, the DSLs evaluation under characteristics that affect the ability to write in a language, such as its expressiveness, is still an insufficiently investigated area, and few are presented in studies. On the other hand, when these evaluations are proposed, approaches specifically directed to the languages under study are used, that is, they do not show an analysis based on metrics that can validate any type of DSL. Considering this problem, the present study aimed to develop an approach through a case study with SQL for evaluations of DSLs based on their grammar, athwart the proposition, and application of quantitative metrics related to the characteristic of expressiveness. The results of the study indicate that the expressiveness gain was observed mainly by the significant increase of 160 new terminals in the SQL version of 1999. These new terminals are related to categories such as data types, commands, expressions, and exceptions that were added to the language. This work can help people interested in the area of DSLs and languages in general, to evaluate the expressiveness quality of their language under study, with a new vision based on the analysis of the grammar developed.

Keywords: language expressiveness; Domain-Specific Language evaluation; grammar-based metrics.

Agradecimentos

Primeiramente agradeço a Deus, por ter me concedido sabedoria para realização deste estudo, e por toda sua misericórdia e bondade para comigo.

Aos meus familiares, especialmente a minha mãe, meu pai, minhas tias e minha avó materna, por terem me educado e sempre me apoiado nas minhas escolhas com todo carinho e amor. Vocês foram meu suporte nos momentos mais difíceis e me fizeram chegar até aqui.

Ao meu irmão e amigos próximos, toda minha gratidão pelos incentivos, palavras e companheirismo durante essa caminhada, vocês tornaram tudo mais leve.

Ao meu orientador, Franklin Ramalho, pela paciência, conselhos e orientação que foi fundamental para a realização deste trabalho. A trajetória foi longa e sem a sua compreensão eu não teria conseguido finalizar.

Agradeço ainda à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro ao longo dessa jornada.

Conteúdo

Lista de Símbolos

DSL - *Domain-Specific Language*

GPL - *General-Purpose Language*

SQL - *Structured Query Language*

GLC - *Gramática Livre de Contexto*

LW - *Language Workbench*

LLC - *Linguagem Livre de Contexto*

BNF - *Backus Normal Form*

SGBDs - *Sistemas de Gerenciamento de Bancos de Dados*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Linguagens Específicas de Domínio, em inglês conhecidas como *Domain-Specific Languages* (DSLs) visam facilitar o desenvolvimento de artefatos de *software* por meio de abstrações e notações especializadas [30]. Muitas atividades da engenharia de *software* estão utilizando cada vez mais DSLs, inclusive para projetar e verificar regras estruturais (*e.g.* Schmitt *et al.* [48]; Moha e Guéhéneuc [35]). No entanto, os resultados da preliminar revisão sistemática da literatura de Gabriel *et al.* [15] e mais recentemente, do estudo de mapeamento sistemático de Kosar *et al.* [29], apontam preocupações com a implantação dessas linguagens específicas para o uso. As discussões estão relacionadas à construção de DSLs inadequadas devido a avaliação desses tipos de linguagens serem insuficientemente investigadas e pouco apresentadas nos trabalhos.

Em particular, uma DSL mal projetada pode se tornar de difícil utilização para os usuários do domínio. Por essa razão, a avaliação da linguagem é indispensável, e não só abrange as correções de *bugs* relacionados por exemplo, a geração de código, *parsing* mal feito, mas também à avaliação de características importantes, como a expressividade, concisão, integração e desempenho de um artefato DSL [24]. A avaliação mediante diferentes características contribuem para reduzir problemas na implantação, fornecendo visões prévias da qualidade e os benefícios econômicos e de produtividade para as organizações com a introdução da DSL.

Estudos apontam a necessidade de uma abordagem sistemática para avaliar DSLs em virtude do crescente número e aumento de complexidade [27] [55]. Reforçando o que foi mencionado, Albuquerque *et al.* [4] revelam que faltam estudos que se baseiem em processos

quantitativos, com criação de um conjunto de métricas para apoiar análises e comparações objetivas entre DSLs. Em vista disso, os autores realizam um trabalho para analisar e comparar quantitativamente e qualitativamente a usabilidade de duas DSLs usadas em tarefas de manutenção de *software*, a partir de métricas criadas pelos próprios autores.

1.1 Problemática

Nos últimos anos, poucos estudos investigaram sobre a avaliação de linguagens desenvolvidas para domínios específicos [6] [42] [41] [4] [26] [48]. O trabalho de Poltronieri [41] destaca que na literatura esta é uma área recente e apresenta uma preocupação crescente, tendo em vista que engenheiros de *software* estão buscando cada vez mais novos métodos para avaliar a qualidade dos seus produtos. Quando uma DSL é mal projetada, as estimativas de custos de manutenção aumentam, até porque em um programa excessivamente complexo há a possibilidade de ser de difícil compreensão e, consequentemente mais caro de manter [43]. Dessa forma, a qualidade do *software* termina sendo uma preocupação constante de engenheiros de softwares que buscam reduzir custos de manutenção do *software* desenvolvido [41] [43].

Com o crescente número e complexidade de algumas DSLs desenvolvidas, alguns estudos apontam a necessidade de uma abordagem sistemática para avaliações de DSLs [15] [55] [27]. Schmitt *et al.* [48] cita que abordagens quantitativas estão declaradas como um problema em aberto no campo de pesquisa de linguagens específicas de domínio. Estudos como o Power e Malloy [43] e Albuquerque *et al.* [4] indicam uma abordagem utilizando métricas de *software* para ajudar a medir características que são importantes da linguagem. O uso de métricas de *software* tornou-se indispensável para uma boa engenharia de *software* [12]. Então a criação de um conjunto métricas para ajudar na análise quantitativa de DSLs possibilitaria uma comparação objetiva entre DSLs [6] [15] [50].

O problema a ser abordado neste trabalho é a falta de estudos para avaliação quantitativa da característica de expressividade para qualquer tipo de DSL, tendo em vista que as abordagens de avaliação apresentadas na área de estudo são direcionadas especificamente para as linguagens que estão sendo investigadas e são voltadas para medir características como complexidade e usabilidade.

É importante a avaliação de linguagens específicas mediante a característica de expressividade devido ela estar relacionada a amplitude de ideias que podem ser representadas e comunicadas nessa linguagem, ou seja, uma linguagem pode ser dita como expressiva quando ela ajuda a produzir códigos com uma maior variedade e quantidade de ideias de forma clara, natural, intuitiva e concisa [60].

1.2 Objetivos

O trabalho tem como objetivo geral a propositura de uma abordagem com *Structured Query Language* (SQL) para avaliações de DSLs baseada em suas gramáticas, em que foi ilustrada e avaliada por um estudo de caso através da aplicação de métricas quantitativas relacionadas à característica de expressividade.

Considerando o objetivo geral deste estudo, os seguintes objetivos específicos foram definidos:

- Analisar os tipos de mudanças encontradas considerando duas versões de gramáticas da linguagem SQL 92 e 99;
- Propor métricas quantitativas baseadas em gramáticas para avaliar a característica de expressividade da linguagem;
- Aplicar as métricas criadas utilizando as gramáticas de SQL;
- Analisar se a expressividade é superior da linguagem SQL em 1999, quando comparada com a 1992;
- Verificar se a utilização das métricas contribuiu para avaliar a expressividade da linguagem SQL.

Por fim, vale salientar que este trabalho não é direcionado apenas para quem usa gerenciadores de banco de dados ou a própria linguagem SQL, mas sim para todos aqueles interessados na área de avaliações quantitativas de linguagens desenvolvidas para domínios específicos.

1.3 Escopo

O escopo da pesquisa é analisar e avaliar quantitativamente a característica de expressividade realizando um estudo de caso com a linguagem SQL, onde tomamos como base as Gramáticas Livres de Contexto (GLCs) das versões de 1992 e 1999 definidas no Formalismo de Backus-Naur (BNF) [28]. Outros artefatos de especificação léxica e semântica que estão ligados as fases de um compilador não são parte do escopo deste trabalho, no caso apenas GLCs são utilizadas para avaliação da DSL SQL.

1.4 Estrutura do documento

O conteúdo desta dissertação está organizado da seguinte forma:

- no Capítulo 2, são apresentados os conceitos fundamentais para auxiliar no entendimento deste trabalho, incluindo detalhes sobre *Domain-Specific Languages*, gramáticas livres de contexto, tipos de mudanças em gramáticas livres de contexto, e por fim acerca de capacidade de escrita e expressividade;
- no Capítulo 3, é encontrada a metodologia que foi aplicada no desenvolvimento desta pesquisa;
- no Capítulo 4, é mostrada a análise das gramáticas de SQL nas versões de 1992 e 1999;
- no Capítulo 5, são apresentadas as métricas criadas para avaliação de expressividade da linguagem;
- no Capítulo 6, são apresentados os resultados e discussões da aplicação das métricas nas versões de SQL apresentadas no Capítulo 5;
- no Capítulo 7, são apresentados os trabalhos relacionados;
- no Capítulo 8, encontram-se as considerações finais e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Domain-Specific Languages

Domain-Specific Languages (DSLs) são linguagens criadas com o objetivo de expressar instruções para um específico espaço de problema ou domínio [6]. Diferentemente das *General-Purpose Languages* (GPLs), em português conhecidas como Linguagens de Propósito Geral, que são amplamente utilizadas para resolver problemas entre muitos domínios. As DSLs são consideradas como pequenas linguagens, orientadas a problemas e de tarefas específicas, nas quais oferecem apenas um conjunto reduzido de notações e abstrações, além de possuir expressividade bastante limitada. São alguns exemplos de DSLs, HTML [38] para páginas web, XML [65] para codificação de dados, SQL [61] para consultas de bancos de dados, DOT [18] para definir gráficos, VHDL [63] usado para definir circuitos, Gherkin [20] para definir testes funcionais, Dynamix [36] para semântica dinâmica, CircuitFlow [10] para programação de fluxo de dados, Latex [59], Mathematica [64], Verilog [62], entre outras. Em contrapartida, as linguagens como C e Java não podem ser consideradas DSLs porque não se limitam a um único domínio, são linguagens de programação capazes de criar todos os tipos de aplicações.

As DSLs têm sido vastamente utilizadas como ferramentas eficazes para apoiar o trabalho de desenvolvedores (*e.g.* Gurgel [19]; Terra e Valente [56]; Ubayashi *et al.* [57]; Morgan *et al.* [37]) na solução de problemas de forma declarativa, abstrata e concisa, utilizando conceitos e termos que lhe são familiares, eliminando detalhes de implementação. Essas linguagens específicas são classificadas em três categorias diferentes: interna, externa, e por

último as chamadas *Language Workbenches* (LWs) [14].

As chamadas internas são linguagens que usam a infraestrutura de uma linguagem de programação existente, também denominada de linguagem nativa, onde terminam sendo implementadas como bibliotecas e componentes que ampliam a linguagem nativa criando novos tipos de dados, rotinas, procedimentos, entre outros [14]. Alguns exemplos são os vários *frameworks* na linguagem de programação Ruby[1] (especialmente o Ruby on Rails[2]) e Scala[3] e arquivos de configuração em Lua[4].

Por outro lado, as DSLs externas são linguagens que possuem sua própria infraestrutura para etapas de análise (*parsing*), interpretação, compilação e geração de código. Essas linguagens são desenvolvidas como qualquer outra, dispõem de sintaxe e estruturas próprias da linguagem [14]. Como exemplos desse tipo temos SQL[5], AWK[6], makefiles[7], entre outras.

As *Language Workbenches* são ferramentas que permitem desenvolver DSLs (*e.g.* Xtest[8], MPS[9], Racket[10] e Spoofax[11]) juntamente com seu ambiente de desenvolvimento integrado. Fornecem um ambiente de trabalho que tem como representar a definição de um modelo semântico, editar a linguagem específica de domínio e criar uma semântica comportamental de interpretação e geração de código, em que acaba facilitando e reduzindo os custos de desenvolvimento de uma DSL [13].

A estrutura de uma DSL, seja ela interna ou externa, é composta de partes fundamentais em seu processamento. A primeira etapa do processamento é a do modelo de execução de um *script* escrito em uma DSL, essa etapa é semelhante para esses dois tipos de DSLs. Na etapa dois, com a execução do *parsing* do *script* de entrada concluída, o *parsing* de dados e todas as operações extraídas são tratadas. Então, na terceira etapa do processamento é realizada a geração do modelo semântico para representação do domínio da aplicação e produção do roteiro de execução.

[1] https://www.ruby-lang.org/
[2] https://rubyonrails.org/
[3] https://www.scala-lang.org/
[4] https://www.lua.org/
[5] https://www.w3schools.com/sql/
[6] https://wikipedia.org/wiki/AWK
[7] https://wikipedia.org/wiki/Makefile
[8] https://www.eclipse.org/Xtext/
[9] https://www.jetbrains.com/mps/
[10] https://racket-lang.org/
[11] https://www.spoofax.dev/

Em algumas DSLs é capaz de existir uma quarta etapa adicional de geração de código. Esta etapa torna-se necessária caso se precise gerar um *script* de saída para ser executado por outra aplicação ou outro ambiente de execução. Estruturalmente, o que difere uma DSL externa de uma DSL interna é a etapa de *parsing* do *script* de entrada, tendo em vista que em uma DSL externa esta etapa é executada pelos analisadores sintáticos, à medida que as DSLs internas são tratadas pelos analisadores da linguagem nativa na qual a DSL foi implementada.

Já com relação os ganhos substanciais que as DSLs podem prover aos projetos, estão associados especialmente o alto nível de abstração, que elimina detalhes desnecessários na hora da implementação, como também os trechos de código, uma vez que em DSLs são mais claros e concisos; além disto, permitem que as soluções sejam expressas no nível do domínio da aplicação, o que possibilita, por exemplo, que analistas de negócio entendam, validem, otimizem, modifiquem e até mesmo desenvolvam funcionalidades utilizando a DSL; e por fim, proporcionam o aumento na produtividade [6].

Em contrapartida, DSLs oferecem também desvantagens que estão relacionadas à falta de suporte de ferramentas adequadas para o desenvolvimento e falta de abordagens para validação e manutenção da DSL; necessidade de precaução com os custos elevados; juntar e definir todo o conhecimento relevante ao domínio específico; e possível baixo desempenho devido ao aumento dos níveis de abstração e a provável falta de um otimizador do código criado.

2.2 Gramáticas Livres de Contexto

Uma gramática $G = (V, T, P, S)$ constituída por V o conjunto finito de símbolos não terminais ou variáveis, T o conjunto finito de símbolos terminais, P o conjunto de produções, ou regras de produção e S o símbolo não terminal inicial, é definida como uma Gramática Livre de Contexto (GLC) se todas as regras de produções presentes em P, são da forma:

$$\mathrm{A} \rightarrow \alpha$$
$$\text{onde A} \in V \text{ e } \alpha \in (V \cup T)^*$$

Tendo em vista que $A \in V$, então, do lado esquerdo da produção, é possível apenas aparecer uma variável, definida como símbolo não terminal. Entretanto, do lado direito, podem

aparecer quaisquer combinações, já que $\alpha \in (V \cup T)^*$, ou seja, quaisquer combinações entre as variáveis e símbolos terminais [44].

Essa estrutura de gramática possui liberdade em ter uma longa listagem de palavras, e também regras sobre os tipos de palavras que podem ser adicionadas e em qual ordem. Tanto as frases como as palavras, podem ser combinadas por diferentes normas e regras a fim de formar sentenças que dispõem de significado.

As gramáticas livres de contexto geram as conhecidas Linguagens Livres de Contexto (LLCs). Essas linguagens são fundamentais por deterem um universo mais amplo de linguagens e por tratarem de questões típicas de linguagens de programação, como exemplo, tratam questões como as dos parênteses balanceados, construções de blocos e estruturas. Elas são consideradas relativamente simples e possuem eficiência razoável [32].

Considerando o exemplo a seguir, L_1 é uma LLC constituída de expressões aritméticas contendo parênteses balanceados, um operador e dois operandos, sua GLC é definida como $G_1 = (\{S\}, \{+, *, (,), a\}, P, S)$ tal que

$$P = \{S \to S + S$$
$$S \to S * S$$
$$S \to (S)$$
$$S \to a\}$$

Então gera as cadeias $\{a, a+a, a*a, (a+a), (a*a), (a+a)*a...\}$ pertencem a L_1 [44].

2.3 Tipos de Mudanças em Gramáticas Livres de Contexto

Buscando na literatura acerca de outros trabalhos que tratassem sobre tipos de mudanças em gramáticas livres de contexto, foi encontrado o catálogo de seis padrões de refatoração gramatical de Halupka e Kollár [21]. Os autores mostram soluções genéricas para problemas de refatoração em gramáticas. Os seis padrões são chamados de *Unfold*, *Fold*, *Remove*, *Pack*, *Reduce* e *Extend*, eles serão detalhados a seguir.

O padrão *Unfold* tem como objetivo reduzir a contagem de símbolos não-terminais da gramática, bem como a profundidade das árvores de derivação construídas. Como solução de refatoração desse padrão na gramática, é necessário substituir o não-terminal do lado

esquerdo pela sua própria regra de produção em cada regra da gramática que faça uso desse não-terminal do lado direito. Por outro lado, o *Fold* é o padrão inverso de *Unfold*, tem como propósito reduzir o comprimento das regras de produção, tal como reduzir o número de nós filhos diretos para cada nó das árvores de derivação construídas. Para esse caso de refatoração, a solução é substituir do lado direito das regras de produção as ocorrências cujo lado esquerdo é o mesmo não-terminal em outra regra de produção por este não-terminal.

O padrão *Remove*, como o próprio nome sugere, tem como finalidade remover os símbolos não-terminais inacessíveis, assim como suas regras de produção. A solução desse padrão é encontrar alguma sequência de derivação inexistente e remover este não-terminal juntamente com suas regras de produção. E o padrão *Pack*, busca remover o comprimento de regras de produções na gramática, e para isso propõe criar um novo não-terminal para empacotar uma sequência de símbolos existentes do lado direito de alguma(s) regra(s) de produção na gramática. Assim, essa sequência de símbolos na(s) regra(s) deve ser substituída pelo novo não-terminal recém-criado.

Os dois últimos, *Reduce* e *Extend* são padrões opostos. O *Reduce* remove regras de produção duplicadas, reduz a contagem de símbolos não-terminais e reduz a contagem de regras de produção. Como solução de refatoração, esse padrão deve verificar se existem dois não-terminais com conjunto equivalente de regras, se for o caso, um dos não-terminal é removido e todas as suas regras de produção, onde todas as ocorrências do não-terminal removido na gramática deve ser substituído pelo outro não-terminal. Por outro lado, o padrão *Extend* aumenta a contagem de não-terminais e profundidade das árvores de derivação construídas. Esse padrão tem como solução a criação de um novo não-terminal e suas produções, dessa forma ocorre no lado direito de algum não-terminal da gramática a substituição deste novo não-terminal recém-criado na regra de produção.

Na fase inicial desta pesquisa, realizamos investigações sobre quais mudanças ocorrem em gramáticas livres de contexto. Considerando as análises das observações diretas das versões de 1992 [46] e 1999 [47] das gramáticas de SQL, escritas no Formalismo de *Backus-Naur* [28], criamos uma classificação em dois tipos de mudanças, sendo essas:

- *Mudanças diretas*: relacionadas às mudanças de *criação*, *adição* ou *remoção* de terminais, não-terminais e regras de produção na gramática. Essas mudanças apresentam notável impacto quando acontecem, tendo em vista que novas especificações são in-

cluídas ou outras especificações são removidas. Com isso o tamanho da gramática muda significativamente.

- *Mudanças de aperfeiçoamentos*: essas mudanças estão voltadas para manutenção e melhoria na compreensão da gramática. Como exemplos desses tipos de mudanças estão a alteração dos nomes de não-terminais e ajustes nas definições das regras de produção, visando colaborar na evolução da gramática.

Esses dois tipos de mudanças foram detectadas a partir da construção de árvores de derivação de exemplos em SQL de 92 e 99, como também através da análise das regras de produção dessas gramáticas. O material das árvores de derivação e análises das regras de produção pode ser observado no Apêndice A.

A principal diferença da classificação dos dois tipos de mudanças, criadas neste trabalho, para a proposta do catálogo com seis padrões de refatoração gramatical é que os autores Halupka e Kollár [21] apresentam formas para solucionar problemas de refatoração mostrando os padrões, enquanto os tipos de mudanças que encontramos definem de forma simplificada como estão presentes essas mudanças nas gramáticas, bem como o impacto delas.

Comparando os dois tipos de mudanças em gramáticas propostas neste trabalho, *mudanças diretas* e *mudanças de aperfeiçoamento* com os padrões de refatoração de Halupka e Kollár [21] vistos acima, conseguimos evidenciar exemplos de refatoramentos realizados na gramática de SQL 99. No exemplo da Figura 2.1, o tipo de mudança encontrado na regra de produção de <*table definition*> se enquadra como de *aperfeiçoamento* e o padrão de refatoração gramatical utilizado identificamos o *Pack*.

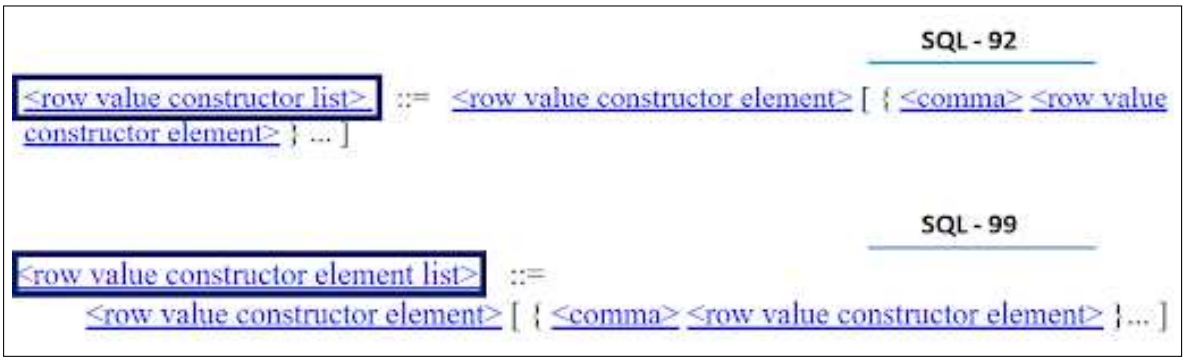
Observamos no exemplo acima que a *mudança de aperfeiçoamento* realizada na gramática foram nas novas definições das regras de produção <*table scope*> e <*global or local*> para pertencer os terminais *GLOBAL*, *LOCAL* e *TEMPORARY*. Por outro lado, o padrão *Pack* foi identificado em comum neste exemplo porque verificamos a redução do comprimento da regra de produção <*table definition*> na gramática com a criação do não-terminal <*table scope*> para empacotar os terminais.

Figura 2.1: Regra de produção <*table definition*>

Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

No exemplo da Figura 2.2 da regra de produção <*row value constructor list*> observamos apenas a *mudança de aperfeiçoamento*, sem a aplicação de nenhum dos tipos de refatoramentos apresentados anteriormente. A *mudança de aperfeiçoamento* encontrada está relacionada à alteração do nome do não-terminal para <*row value constructor element list*> na versão de 1999 da gramática de SQL. Esse tipo de mudança não reflete nenhum impacto na gramática, uma vez que a criação ou remoção de novas especificações não foram feitas.

Figura 2.2: Regra de produção <*row value constructor list*>



Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

2.4 Capacidade de Escrita e Expressividade

Na literatura existem critérios de avaliação de linguagens [45] [49], dentre eles está o critério capacidade de escrita (*writability*), que é entendido como a medida de quão facilmente uma linguagem pode ser usada para desenvolver programas para um determinado domínio de problema escolhido [49]. Quando novas linguagens de programação são projetadas, este critério é considerado como um dos principais para avaliação, uma vez que tornar os programas fáceis de escrever é um dos objetivos mais relevantes para linguagens, e por isso devem dispor de uma boa *writability* [7].

A capacidade de escrita de uma linguagem, é vista também como uma **qualidade** quando esta permite ao programador expressar computações por meio de estruturas claras, corretas, concisas, curtas e de forma rápida [31]. Samimi-Dehkordi, Khalilian e Zamani [45] retratam a *writability* como "a capacidade de dizer o que você quer dizer sem qualquer verbosidade expressiva.".

Existem subcritérios ou características que afetam a capacidade de escrita em uma linguagem, são exemplos deles: simplicidade e ortogonalidade, tipos de dados, construção da sintaxe, suporte para abstração e expressividade [45] [49]. Neste trabalho, nos concentramos em aprofundar o subcritério expressividade, em inglês chamado como *expressiveness* ou *expressivity* ou ainda *expressive power*.

Expressividade, de acordo com o dicionário Dicio [3], é a característica do que é expressivo. Sendo expressivo [2], um adjetivo que define ser claro, significativo, em que se dá a entender, em que há expressão, entre outros. Em computação, uma linguagem é dita como expressiva quando é possível especificar ideias para resolução de problemas de um programa de forma natural, conveniente, apropriada e sem a necessidade de recorrer à artefatos como documentações das peças do software, como manuais do usuário, relatórios de implementação, comentários, anotações e assim por diante [39].

A expressividade auxilia na capacidade de escrita, visto que ajuda linguagens a especificar cálculos de formas convenientes, em vez de complicados. Um exemplo disso é o uso do **count++** , maneira mais conveniente, fácil e curta do que **count = count + 1**. Outro exemplo, é a inclusão do **for** em algumas linguagens (como em Java) para escrever mais facilmente *loops* de contagem do que o uso do **while**, o que também é possível [49].

Logo, a expressividade é considerada uma habilidade da linguagem quando permite expressar uma grande quantidade de "computações" em algumas poucas linhas de código. Por outro lado, a carência em expressividade nas linguagens (como em Assembly) tornam operações relativamente simples em algo complexo, por exigir maior quantidade de "computações", ou seja, muitas linhas de código.

Dessa forma, quando novos operadores são acrescentados na linguagem, por exemplo, isso pode ser um indicativo do aumento de expressividade desta linguagem. Considerando que eles podem estar relacionados a comandos, tipos de dados, funções que visam ajudar o usuário na escrita de códigos com menor complexidade e maior clareza, concisão e facilidade.

Expressividade é apontada ainda como uma das principais características de DSLs em Oliveira *et al.* [39] e Haugen e Mohagheghi [22]. Essas linguagens são conhecidas por apresentar grande poder expressivo, onde usam notações de alto nível e abstrações apropriadas direcionadas para um domínio de problema específico [58] [55]. Os conceitos que são expressados para um único domínio, torna essas DSLs eficientes, sobretudo no que se refere à linguagem ser lida e aprendida pelos especialistas de domínio [9]. Especialistas de domínio são geralmente pessoas não desenvolvedoras com nenhum ou pouco conhecimento em programação mas, possuem grande conhecimento na área para a qual a DSL está sendo projetada. Dessa maneira, segundo Oliveira *et al.* [39], em razão das DSLs normalmente serem pequenas, abstratas e expressivas, os especialistas conseguem facilmente ler os programas e aprender essas linguagens para especificar os programas com maior eficiência, ou em outras palavras, com pouco tempo gasto.

Veja nos Códigos 2.1 e 2.2 a evolução em expressividade da linguagem SQL no que se refere ao suporte de novos tipos de dados. No Código 2.1, mostra a criação de uma tabela de carros que tem como colunas o nome, marca, id_carro, ano e ar_condicionado. Na linguagem de 92 para representar que a coluna ar_condicionado recebe os valores TRUE e FALSE, isso era apenas possível utilizando o tipo de dado BIT, sendo 1 equivalente a verdadeiro e 0 falso. Por outro lado, observamos no Código 2.2 do ano de 99 que o tipo BOOLEAN foi incluído na linguagem, tornando-se possível utilizar em consultas também os valores booleanos TRUE e FALSE, e desconhecido com o UNKNOWN.

Código 2.1: Exemplo SQL 92 utilizando o tipo de dado BIT

```
1  CREATE TABLE Carros(
2          nome VARCHAR (30),
3          marca VARCHAR (20),
4          id_carro INT IDENTITY,
5          ano INT NOT NULL,
6          ar_condicionado BIT NOT NULL);
7
8  INSERT INTO Carros (
9          nome,
10         marca,
11         ano,
12         ar_condicionado)
13 VALUES('Ecosport', 'Ford', 2016, 1)
```

Código 2.2: Exemplo SQL 99 utilizando o tipo de dado BOOLEAN

```
1  CREATE TABLE Carros(
2          nome VARCHAR (30),
3          marca VARCHAR (20),
4          id_carro INT IDENTITY,
5          ano INT NOT NULL,
6          ar_condicionado BOOLEAN NOT NULL);
7
8  INSERT INTO Carros (
9          nome,
10         marca,
11         ano,
12         ar_condicionado)
13 VALUES('Ecosport', 'Ford', 2016, TRUE)
```

Já com relação aos Códigos 2.3 e 2.4, veja como ficou a escrita de uma mesma consulta em SQL com e sem a estrutura de dados ARRAY. Uma tabela chamada filmes está sendo ordenada por gênero, de forma que o gênero ação é o último a ser avaliado. O Código 2.4 usando a estrutura de dados ARRAY é visto melhorias, sobretudo na quantidade menor de número de linhas para realizar a operação em SQL.

Código 2.3: Exemplo sem a estrutura de dados ARRAY

```
1  SELECT * FROM filmes ORDER BY CASE genero
2  WHEN 'ACAO' THEN 0
3  WHEN 'AVENTURA' THEN 1
4  WHEN 'COMEDIA' THEN 2
5  WHEN 'DRAMA' THEN 3
6  WHEN 'TERROR' THEN 4
7  WHEN 'FANTASIA' THEN 5
8  WHEN 'ROMANCE' THEN 6
9  WHEN 'FICCAO_CIENTIFICA' THEN 7
10 WHEN 'MUSICAL' THEN 8
11 WHEN 'DOCUMENTARIO' THEN 9
12 WHEN 'ANIMACAO' THEN 10
13 WHEN 'INFANTIL' THEN 11
14 END ASC, avaliacao DESC;
```

Código 2.4: Exemplo usando a estrutura de dados ARRAY

```
1  SELECT * FROM filmes ORDER BY
2  ARRAY_POSITION(ARRAY['ACAO', 'AVENTURA','COMEDIA','DRAMA',
3  'TERROR','FANTASIA','ROMANCE','FICCAO_CIENTIFICA',
4  'MUSICAL','DOCUMENTARIO', 'ANIMACAO', 'INFANTIL']::VARCHAR[],
       genero), avaliacao DESC;
```

Nos Códigos 2.5 e 2.6 consideramos o uso das estruturas de repetição **for** e **while** também na linguagem SQL, ambas podem ser empregadas para realizar consultas semelhantes, porém dependendo de determinadas operações e o contexto aplicado essa escolha da estrutura pode impactar na expressividade, complexidade, performance, entre outros. Em SQL 92 somente o laço **for** existia para realizar iterações, sendo esse usado apenas quando sabe-se o número de iterações com antecedência (veja o Código 2.5). Já em SQL 99, foi acrescentado o construtor **while**, permitindo escrever repetições em outro formato, onde um certo trecho de programa é executado ENQUANTO uma certa condição for verdadeira (veja o Código 2.6).

Código 2.5: Exemplo usando a estrutura FOR

```
1  CREATE OR REPLACE FUNCTION fatorial (n integer) RETURNS integer AS
2
3  DECLARE
4          fat integer := 1;
5          i integer;
6  BEGIN
7          IF n<2 THEN
8                  RETURN fat;
9          END IF;
10         FOR i in 2..n LOOP
11                 fat := fat*i;
12         END LOOP;
13         RETURN fat;
14 END;
```

Código 2.6: Exemplo usando a estrutura WHILE

```
1  CREATE OR REPLACE FUNCTION fatorial (n integer) RETURNS integer AS
2
3  DECLARE
4          fat integer := 1;
5          i integer := 2;
6  BEGIN
7          WHILE (i <=n) LOOP
8                  fat := fat*i;
9                  i := i+1;
10         END LOOP;
11         RETURN fat;
12 END;
```

Capítulo 3

Metodologia

No presente capítulo encontra-se a metodologia aplicada no desenvolvimento desta pesquisa. Considerando a importância de desenvolver estudos na área de avaliação de linguagens específicas de domínio, o método estudo de caso foi escolhido como adequado para ilustrar a abordagem desta pesquisa. De acordo com Patton [40], o método tem como finalidade reunir informações detalhadas e sistemáticas sobre um fenômeno. Nesse sentido, o estudo de caso tem o propósito ser um estudo profundo e exaustivo de um ou múltiplos objetos, de maneira que se permita o seu amplo e detalhado conhecimento [17].

Nos primeiros passos desta pesquisa realizamos um levantamento bibliográfico de caráter sistemático com o objetivo de identificar artigos na literatura relacionados a abordagens de avaliações quantitativas de DSLs, e procuramos também estudar e analisar mais profundamente o problema de pesquisa que está associado à falta de estudos para avaliação quantitativa sob características que afetam a capacidade de escrita em uma linguagem, como exemplo a expressividade. Por outro lado, quando essas abordagens de avaliações são propostas, elas são direcionadas especificamente para as linguagens que estão sendo investigadas, ou seja, não mostram uma análise baseada em métricas que possam validar qualquer tipo de DSL e as características priorizadas nas avaliações são as de complexidade e usabilidade.

Em seguida, com o intuito de definir uma abordagem para avaliação quantitativa de DSLs que conseguisse ser utilizada por qualquer tipo de DSL, escolhemos a linguagem SQL e duas gramáticas BNF dos anos de 1992 e 1999 para realizarmos análises dos terminais, não-terminais e regras de produção. Inicialmente produzimos um estudo onde foram montados alguns exemplos de códigos de consultas em SQL 92 e 99 com construções de árvores de de-

rivação para que pudéssemos identificar os tipos de mudanças ocorridas considerando as duas linguagens nos anos distintos, esse material pode ser observado no Apêndice A juntamente com as análises das regras de produção entre essas mesmas gramáticas dando importância aos exemplos desses códigos que foram elaborados.

Os artefatos acima produzidos na fase inicial do trabalho abriram os caminhos para a identificação dos tipos de mudanças que podem ser diretas e de aperfeiçoamento (detalhadas na Seção 2.3) em GLCs; contribuindo ainda para criação das métricas quantitativas para mensurar o aumento da característica de expressividade em linguagens específicas de domínio.

As quatro métricas criadas foram desenvolvidas com base em observações diretas detectadas a partir das duas versões das gramáticas de SQL, onde as seções relacionadas aos tipos de dados, caracteres especiais e operadores tornaram-se importantes para construção de tais métricas. Essas seções das gramáticas apontam mudanças que consequentemente impactam no aumento de expressividade da linguagem SQL. Além do mais, as tabelas de referências cruzadas das palavras-chave e das regras de produção dessas gramáticas também foram utilizadas e consideradas essenciais para as observações, uma vez que elas apontam os terminais e não-terminais das gramáticas e as regras de produção usadas por cada um deles.

Com relação à coleta de dados, a quantidade de *keywords*, não *keywords* e regras de produção foram calculadas de forma manual para realizarmos uma análise das diferenças entre as versões das gramáticas de SQL 92 e 99, porém para executar a checagem dos valores encontrados utilizamos a ferramenta Excel [11] para confirmar o que foi calculado. No entanto, existem ferramentas como exemplo do *yacc* [51] e *bison* [52] que realizam a contagem de terminais e não-terminais de gramáticas. Outras ferramentas como exemplo xText [54], Flex/ Lex [53], JFlex [1] podem ser usadas para garantir que a gramática está de acordo e definida corretamente no formalismo BNF.

A avaliação do trabalho se deu com a aplicação das quatro métricas criadas sobre as duas versões de SQL, onde três delas medem a estrutura dos terminais da gramática, enquanto a outra métrica mede a relação dos terminais e não-terminais da gramática. Em conclusão, foi feita uma análise sobre estes resultados no Capítulo 6 para apresentar os ganhos de expressividade da linguagem SQL comparando as versões de 92 e 99.

Capítulo 4

Análise das Gramáticas SQL 92 e SQL 99

Para construção das métricas e análises dos resultados que serão apontadas neste trabalho foram utilizadas as gramáticas de SQL – *Structured Query Language* – dos anos de 1992 [46] e 1999 [47] definidas no Formalismo de Backus-Naur [28] (BNF, do inglês *Backus-Naur Form* ou *Backus Normal Form*). Essas gramáticas foram escolhidas em virtude de estarem facilmente disponíveis na internet e pela linguagem de SQL ser um exemplo de DSL amplamente utilizada no mercado, além de que essas versões apresentam tamanho e complexidade satisfatória para os estudos realizados. Nada obstante, o trabalho pode ser aplicado com qualquer linguagem em qualquer janela de tempo de versões de suas gramáticas.

SQL é conhecida como uma linguagem padrão utilizada para interagir com bancos de dados relacionais, os quais são baseados no modelo relacional e tem como objetivo representar dados em tabelas de forma intuitiva e direta. Alguns dos principais bancos de dados do mercado que usam SQL, como Oracle[1], MySQL[2], MariaDB[3], PostgreSQL[4], Firebird[5], Microsoft SQL Server[6], entre muitos outros. Vale destacar que o modelo SQL usado não é o mesmo empregado por alguns Sistemas de Gerenciamento de Bancos de Dados (SGBDs) presentes no mercado, uma vez que existem outros bancos de dados baseados no modelo não-relacional, hierárquico, de rede e orientado a objetos. Além do mais, o padrão SQL usado como estudo de caso desta pesquisa não se aplica para outras especificidades criadas

[1] https://www.oracle.com/

[2] https://www.mysql.com/

[3] https://mariadb.org/

[4] https://www.postgresql.org/

[5] https://firebirdsql.org/

[6] https://www.microsoft.com/sql-server/

a partir da linguagem SQL, como exemplo da PL/SQL[7] que é uma linguagem procedural da Oracle que estende a linguagem SQL.

Existem comandos muito utilizados em SQL, dentre alguns principais estão o *SELECT* que tem como finalidade consultar/ buscar os dados de uma tabela em um banco de dados. O *INSERT* que tem como objetivo inserir novas linhas na tabela. O *UPDATE* que atualiza um ou mais dados armazenados em uma tabela. Por fim, o *DELETE* é utilizado para que uma ou mais linhas sejam excluídas de uma tabela de banco de dados.

Outros comandos como *JOIN*, *LIKE*, *GROUP BY* e *ORDER BY* fazem outros tipos de buscas melhores e mais elaboradas em SQL. E além desses, existem outros muitos comandos que são usados na linguagem.

No que diz respeito as duas gramáticas de SQL 92 e SQL 99, ambas são compostas por seções e duas tabelas de referências. Logo de início expõem uma seção principal intitulada como *Key SQL Statements and Fragments*, onde as estruturas-chave da linguagem são mostradas seguidas das regras de produção que estão associadas.

Logo em seguida estão apresentadas todas as seções que possuem nas duas gramáticas, juntamente com uma breve descrição do que se trata cada uma delas. Vale destacar que as seções mais utilizadas e importantes para os estudos feitos neste trabalho foram a *Basic Definitions of Characters Used, Tokens, Symbols, Etc.* e *Data Types*, porque através das observações das mudanças feitas dessas seções conseguimos verificar o aumento da expressividade na linguagem SQL.

- *Identifying the version of SQL in use*: inclui as regras de produção relacionadas as definições de versões SQL, linguagens suportadas (Ada, C, COBOL, Fortran, MUMPS, Pascal, PL/I), pacotes, entre outras.

- *Basic Definitions of Characters Used, Tokens, Symbols, Etc.*: contém as regras de produção que mostram os tipos de terminais que são caracteres especiais, operadores, pontuações, letras, números e palavras-chave na linguagem.

- *Literal Numbers, Strings, Dates and Times*: abrange as regras de produção de como expressar literais numéricos em SQL, além de strings, datas e horas.

[7] https://www.oracle.com/br/database/technologies/appdev/plsql.html

- *Data Types*: mostra quais são os tipos de dados suportados na linguagem. Algumas dessas regras de produção são dos tipos bit, numérico, intervalo, data e hora, entre outras.

- *Literals*: mostra os tipos de literais da linguagem, bem como suas especificações de regras de produção para caracteres nacional, hexadecimal, boolean, binário, entre outros.

- *Constraints*: expressa quais são as regras de produção de restrições em SQL para usar nos dados em uma tabela. Alguns exemplos das regras apresentadas são referentes a *UNIQUE* e *PRIMARY KEY*.

- *Search Condition*: engloba as regras de produção para definir condições de busca em SQL, são vistos para alguns predicados como *BETWEEN*, *IN*, *LIKE*, *NULL*, *MATCH*, entre outros.

- *Queries*: seção que mostra as especificações das regras de produção para realizar consultas em SQL, são exemplos apresentados as cláusulas *WHERE*, *SELECT*, *JOIN*, *GROUP BY*, *HAVING*, entre outras.

- *Query expression components*: apresenta as regras de produção referentes aos componentes que podem ser utilizados em expressões de consultas. Cláusulas como *CASE*, *WHEN*, *ELSE* são vistas nessa seção.

- *More about constraints*: mais regras de produção de restrições, apontando o exemplo de restrição *FOREIGN KEY*, entre outras.

- *Module contents*: define as regras de produção de uso de módulos em SQL, que consistem em um grupo de procedimentos. As regras de especificação do *CURSOR*, além do *ORDER BY* são mostradas nessa seção.

- *SQL Procedures*: apresenta o conjunto de regras de produção das especificações de como usar procedures.

- *SQL Schema Definition Statements*: mostra as regras de produção de operações com esquemas.

- *SQL Data Manipulation Statements*: aponta as regras de produção de especificação dos comandos *FETCH*, *OPEN*, *CLOSE*, *DELETE*, *UPDATE*, *INSERT*, entre outros.

- *Connection Management*: mostra as regras de produção de uso do comando *CONNECT*, bem como as suas operações possíveis na linguagem.

- *Session Attributes*: mostra um conjunto de regras de produção para especificar sessões na linguagem, como exemplo especifica a regra de <*set local time zone statement*> para identificar o fuso horário da sessão do usuário, e além disso define também outras regras com as operações permitidas na linguagem SQL para essa regra de produção. Além das regras de sessões, na gramática de 99, observamos também que exibe as regras de comandos como *IF*, *ELSE*, *ELSEIF*, *LOOP*, *WHILE*, *FOR*, entre outras.

- *Dynamic SQL*: apresenta as regras de produção que descrevem como efetuar consultas dinâmicas em SQL, como exemplo cita as especificações do comando *EXECUTE*.

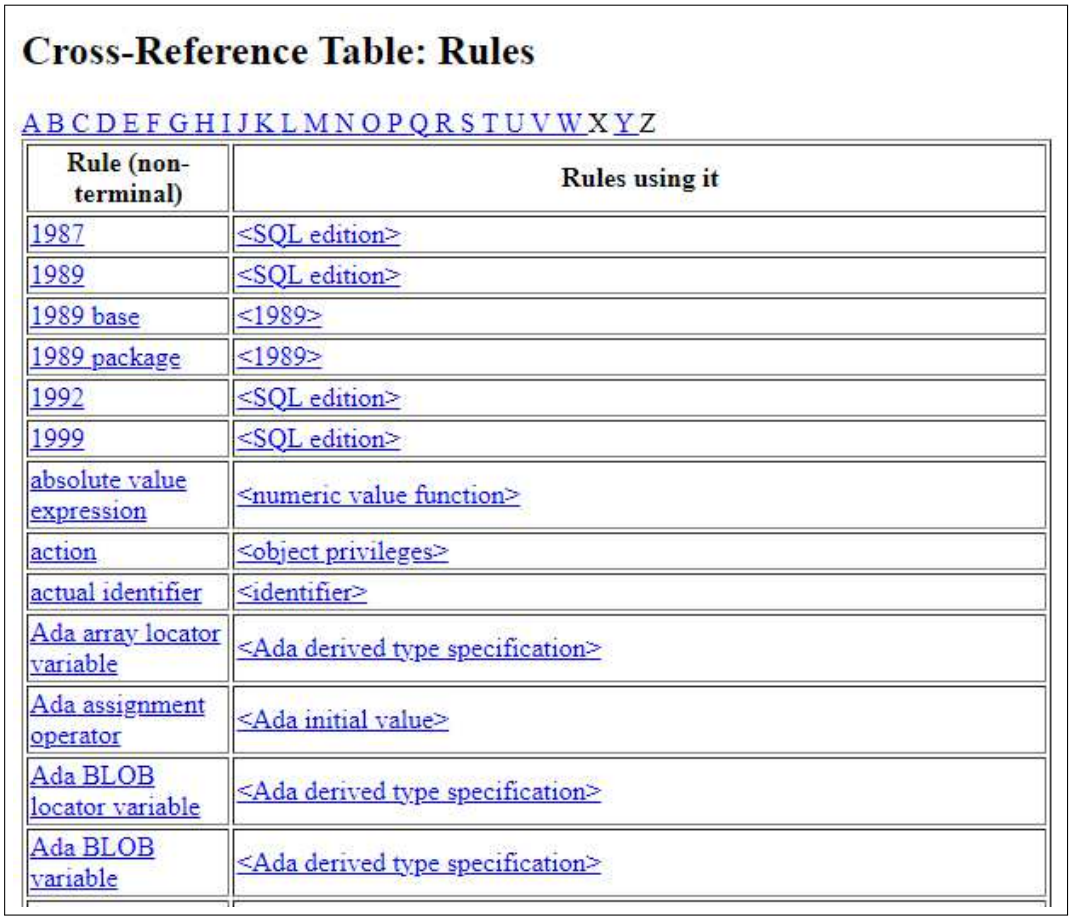
Já com relação as seções que a gramática de SQL 92 tem a mais que SQL 99, observamos unicamente a seção de *SQL Module*, e mesmo assim as regras de produção definidas nessa seção em 92 estão também na gramática de 99. A única diferença é que as regras de produção de *SQL Module* foram transferidas para a seção de *Literal Numbers, Strings, Dates and Times* na gramática de 99, passando a seção de *SQL Module* não existir mais. Essa mudança ocorrida na gramática evidencia um tipo de *mudança de aperfeiçoamento*, visando apenas reorganizar as regras de produção na nova gramática, não apresentando grandes impactos em seu tamanho.

Por outro lado, a gramática de SQL 99 apresentou mudanças significativas nas seções quando comparada com a versão de SQL 92, onde verificamos que novas seções *SQL Schema Manipulation Statements*, *SQL Control Statements* e *Transaction Management* foram adicionadas. A seção *SQL Control Statements* definiu novas regras de produção para a gramática SQL 99 que não existiam nenhuma delas na versão de 92. Além disso, as outras duas seções *SQL Schema Manipulation Statements* e *Transaction Management* incrementaram novas regras de produção em SQL 99, porém outras regras também dessas seções, já pertenciam na gramática de 92 e foram reorganizadas nessas novas seções em SQL 99. Vale ressaltar que essas mudanças ocorridas na gramática são do tipo de *mudanças diretas*, onde a criação de novas regras de produção tiveram forte impacto no tamanho da gramática.

Essas mudanças encontradas nas seções, especialmente da gramática de SQL 99, forneceu indicativos na evolução da expressividade da linguagem SQL com o passar dos anos, uma vez que esses novos não-terminais e terminais criados, adicionados ou removidos da gramática podem ter melhorado a linguagem na capacidade de expressar problemas de forma mais simples e eficiente.

Nas gramáticas de SQL 92 e SQL 99, além das seções que foram detalhadas acima, são mostradas também duas tabelas de referências cruzadas. A primeira tabela *Cross-Reference Table: Rules*, mostra todos os não-terminais da gramática por ordem alfabética seguidas de todas as regras da gramática que utilizam cada um dos não-terminais. Veja a Figura 4.1 abaixo para entender melhor como é definida a tabela em questão.

Figura 4.1: Cross-Reference Table: Rules



Cross-Reference Table: Rules

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Rule (non-terminal)	Rules using it
1987	<SQL edition>
1989	<SQL edition>
1989 base	<1989>
1989 package	<1989>
1992	<SQL edition>
1999	<SQL edition>
absolute value expression	<numeric value function>
action	<object privileges>
actual identifier	<identifier>
Ada array locator variable	<Ada derived type specification>
Ada assignment operator	<Ada initial value>
Ada BLOB locator variable	<Ada derived type specification>
Ada BLOB variable	<Ada derived type specification>

Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

A segunda tabela *Cross-Reference Table: Keywords* mostrada na Figura 4.2, engloba todos os terminais da gramática, conhecidos também como *keywords* da linguagem. Essa tabela também é estruturada em ordem alfabética seguidas de todas as regras de produção da gramática que dispõem cada um desses terminais.

Figura 4.2: Cross-Reference Table: Keywords

Cross-Reference Table: Keywords

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Keyword	Rules using it
ABS	<absolute value expression> <interval absolute value function> <non-reserved word>
ABSOLUTE	<fetch orientation> <reserved word>
ACTION	<referential action> <reserved word>
ADA	<language name> <non-reserved word>
ADD	<add attribute definition> <add column definition> <add column scope clause> <add domain constraint definition> <add original method specification> <add overriding method specification> <add table constraint definition> <reserved word>
ADMIN	<grant role statement> <non-reserved word> <revoke role statement> <role definition>
AFTER	<reserved word> <trigger action time>
ALL	<all> <constraint name list> <disconnect object> <non-join query expression> <non-join query term> <object privileges> <reserved word> <set quantifier> <transforms to be dropped>

Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

Analisando as duas tabelas nas gramáticas de SQL 92 e SQL 99, verificamos que ambas tabelas da versão de 99 sofreram um relativo crescimento em sua extensão, ou seja, tanto os terminais como os não-terminais aumentaram significativamente nessa gramática. Contudo, a tabela *Cross-Reference Table: Rules* da Figura 4.1 é que apresenta maior extensão, independente da versão da gramática, SQL 92 ou SQL 99.

Capítulo 5

Métricas para Avaliação de Expressividade em Gramáticas

Nesta seção serão apresentadas as análises e observações realizadas nas gramáticas de SQL 92 e 99 que contribuíram na elaboração de métricas para avaliação de expressividade da linguagem através de gramáticas.

5.1 Métricas de Expressividade

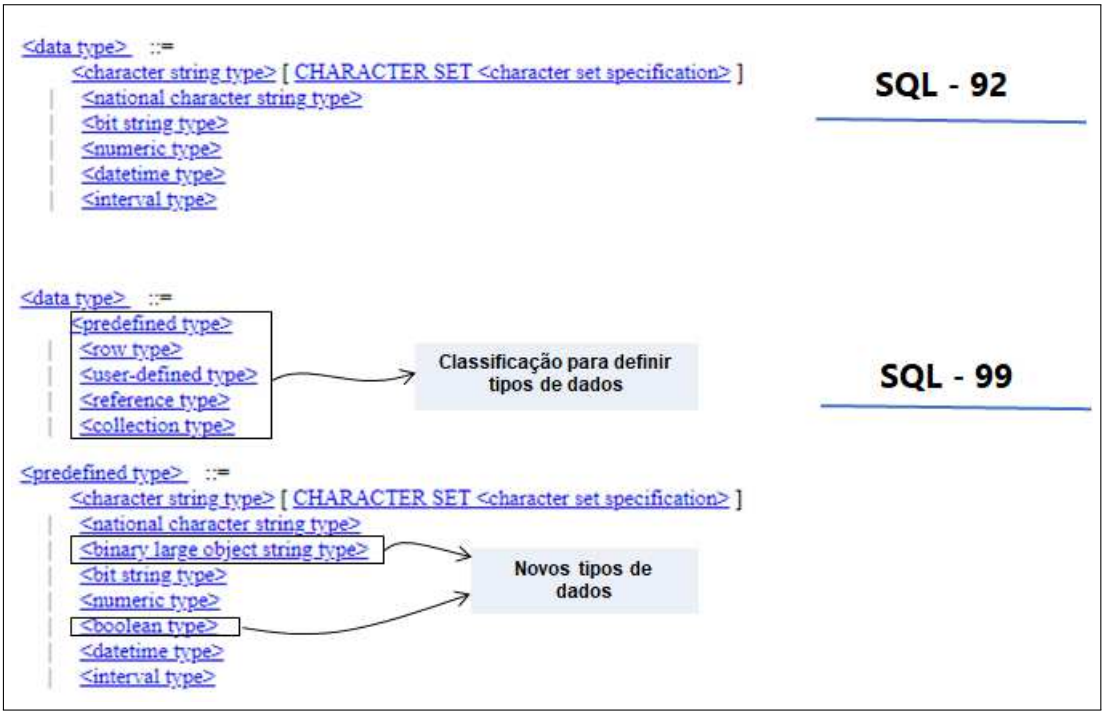
Por meio de observações e comparação da regra de produção <*datatype*> nas gramáticas de 92 e 99 em estudo, verificamos que ocorreram duas mudanças importantes, uma relacionada à classificação para definir tipos de dados e a outra relacionada à adição de novos tipos de dados, conforme ilustrado na Figura 5.1. A definição dos tipos de dados buscou separar, por exemplo, tipo de coleções (*arrays*) de tipos predefinidos (*character*, *numeric*, *bit*, *boolean*, entre outros). Este refatoramento serviu para melhorar apenas a compreensão da gramática, sem nenhuma mudança expressiva na linguagem SQL. Por outro lado, o não-terminal <*pre-defined type*> (ver Figura 5.1), cabeça de uma regra de produção, mostra os novos tipos de dados <*binary large object string type*> e <*boolean type*> criados em SQL 99. Uma maneira de expressar formalmente esta regra, é vista abaixo:

$$r ::= z_1 \mid z_2 \mid z_3 \mid \cdots \mid z_n \tag{5.1}$$

Sendo r o não terminal da cabeça da regra de produção e sendo $z_1 \mid z_2 \mid z_3 \mid \cdots \mid z_n$ todos terminais ou não-terminais do corpo da regra de produção.

Veja que r é um não-terminal, assim como <*predefined type*>. Além disso, não-terminais por exemplo, <*character string type*> e <*national character string type*> (ver Figura 5.2 e Figura 5.3), podem ainda ser representados como $z_2 ::= y_1 \mid y_2 \mid y_3 \mid \cdots \mid y_n$ e $z_3 ::= k_1 \mid k_2 \mid k_3 \mid \cdots \mid k_n$, onde z_2, <*character string type*>, e z_3, <*national character string type*>, expandem também seus terminais e não-terminais nas regras.

Figura 5.1: Comparação da regra de produção <*datatype*>



Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

A partir da evolução da gramática de SQL 92, novos tipos de dados foram adicionados em SQL 99, por exemplo o não-terminal <*predefined type*> passou a ter 08 (oito) tipos (<*character string type*>, <*national character string type*>, <*binary large object string type*>, <*bit string type*>, <*numeric type*>, <*boolean type*>, <*datetime type*> e <*interval type*>), onde 02 (dois) são novos. Na SQL 92 por sua vez, existiam 06 (seis) tipos de dados (<*character string type*>, <*national character string type*>, <*bit string type*>, <*numeric type*>, <*datetime type*> e <*interval type*>). Levando em consideração os formalismos acima apresentados e o exemplo da regra de produção <*predefined type*> na adição de novos tipos, é possível definir

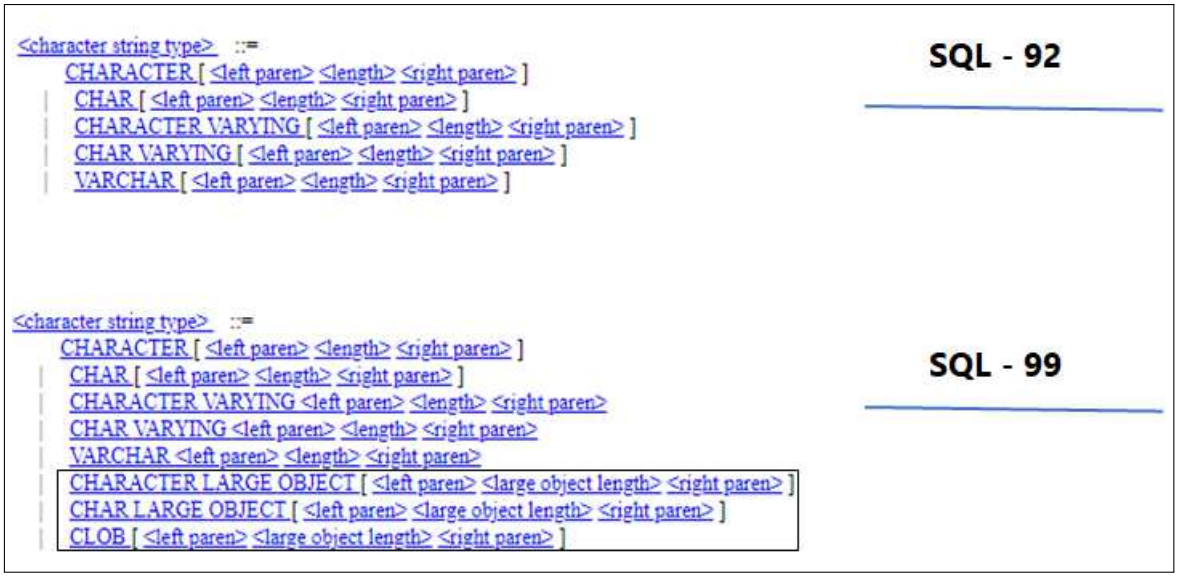
agora r, z_2 e z_3 da seguinte forma:

$$
\begin{aligned}
r &::= z_1 \mid z_2 \mid z_3 \mid \cdots \mid z_n \mid z_{n+1} \\
z_2 &::= y_1 \mid y_2 \mid y_3 \mid \cdots \mid y_n \mid y_{n+1} \\
z_3 &::= k_1 \mid k_2 \mid k_3 \mid \cdots \mid k_n \mid k_{n+1}
\end{aligned}
\tag{5.2}
$$

Os novos tipos adicionados, por exemplo em <*character string type*> e <*national character string type*> em 1999 (ver Figura 5.2 e Figura 5.3), são representados acima como y_{n+1} e k_{n+1}, onde $n+1$ expressa os vários terminais e/ ou não-terminais que podem ser acrescentados em regras anteriormente produzidas. No caso, se y_{n+1} e k_{n+1} forem não-terminais, novas regras serão formadas e consequentemente outras evoluções da gramática são realizadas:

$$
\begin{aligned}
y_{n+1} &::= w_1 \mid w_2 \mid \cdots \mid w_n \\
k_{n+1} &::= q_1 \mid q_2 \mid \cdots \mid q_n
\end{aligned}
\tag{5.3}
$$

Figura 5.2: Comparação do não-terminal <*character string type*>



Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

Com o exemplo acima, observamos que a evolução da linguagem ocorreu pela adição dos novos tipos de dados em regras como <*predefined type*>, <*character string type*> e

Figura 5.3: Comparação do não-terminal <*national character string type*>

Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

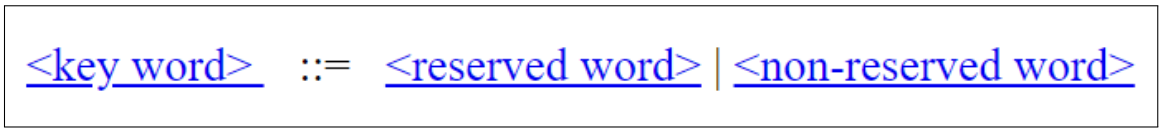
<*national character string type*> na gramática de 99 (rever Figuras 5.1, 5.2 e 5.3), consideramos este fato nas novas investigações a seguir, onde contribuiu na criação de métricas para calcularmos a expressividade.

Nas gramáticas em estudo, as *keywords*, em português palavras-chave, são representadas em letras maiúsculas e são classificadas em dois grupos (ver Figura 5.4): palavras reservadas (*reserved word*) e não reservadas (*non-reserved word*). *Reserved words* segundo a documentação da IBM [25], são todas palavras especiais que executam operações em SQL, são exemplos dessas palavras o *INSERT* ou *CREATE*; elas não podem ser utilizadas como identificadores na linguagem, ou seja, não podem nomear bancos de dados, tabelas, colunas ou quaisquer outros elementos de banco de dados. Por outro lado, *non-reserved words*, também de acordo com a IBM [25], são palavras que apenas em contextos específicos podem apresentar algum significado especial, por exemplo ao usar as palavras não reservadas *UPPER* ou *LOWER* com o objetivo de retornar o sobrenome de pessoas em letras maiúsculas ou minúsculas em tabelas específicas. Esse grupo de palavras não reservadas também pode ser usado como identificadores. Diferentemente das *reserved words*, identificadores são conhecidos segundo Microsoft [33] como o nome do elemento de banco de dados, podendo ser tabelas, exibições, colunas, índices, gatilhos, procedimentos, restrições e regras. Usando o

exemplo anterior, as palavras *UPPER* ou *LOWER* podem ser atribuídas como nome de uma tabela em SQL, assim como todas as outras palavras não reservadas definidas na gramática.

Com base em tudo que foi exposto anteriormente, percebemos que a adição das regras de novos tipos de dados impacta diretamente no número de palavras-chave na gramática. Logo constatamos que a expressividade poderia ser medida através do aumento de palavras-chave quando duas versões de gramáticas da linguagem são comparadas.

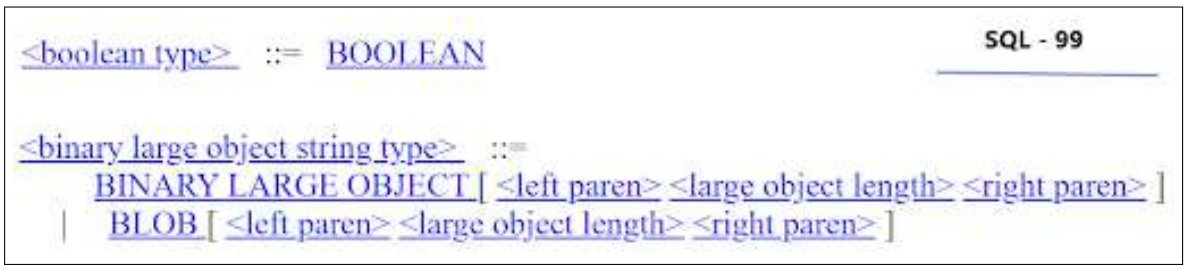
Figura 5.4: Regras de produção <*key word*>



Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

Podemos notar nas Figuras 5.2, 5.3 e 5.5, que as regras <*character string type*>, <*national character string type*>, <*boolean type*> e <*binary large object string type*> evidenciaram o padrão da criação de novas *keywords* ao definir um novo tipo de dado. A Figura 5.2 mostra a comparação da regra de produção de <*character string type*> e as *keywords* em maiúsculos *LARGE*, *OBJECT* e *CLOB* criadas para definir os novos tipos na linguagem SQL 99. Na regra <*national character string type*>, vista na Figura 5.3, foi adicionada a nova *keyword NCLOB*. Por fim, na Figura 5.5 vejamos as regras e novas *keywords* criadas em <*binary large object string type*> e <*boolean type*>, sendo estas *BOOLEAN*, *BINARY* e *BLOB* que estendem a versão anterior da gramática SQL 92.

Figura 5.5: Regras de produção <*binary large object string type*> e <*boolean type*>



Fonte: Gramática BNF – SQL 1999 [47]

Com adição das palavras-chave como os *CLOBs*, *NCLOBs* e *BLOBs* na gramática de

99, verificamos que a linguagem SQL passou a fornecer uma maneira nova de armazenar quantidades extraordinariamente grandes de caracteres usando o *CLOB*, caracteres nacionais através do *NCLOB* e binários com *BLOB*, sendo capazes de armazenar até 4 gigabytes desses dados. Esses são exemplos que evidenciam a evolução da linguagem a partir da criação de novos tipos de dados.

Assim, definimos a **Métrica I – Número de terminais que são *keywords*** para calcular a expressividade da linguagem considerando as palavras-chave da gramática, visto que a adição de regras para novos tipos de dados na gramática mostra diretamente a relação também da criação de novas *keywords*.

Dessa forma, sendo *KeywordsTerminalNumber* o número total de terminais do tipo *Keywords* e sendo *ReservedWordNumber* e *NonReservedWordNumber* o número das palavras-chave reservadas e não reservadas da linguagem. A Métrica I é calculada pelo somatório do número de *reserved words* e *non-reserved words*.

Métrica I – Número de terminais que são *keywords*

$$KeywordsTerminalNumber = ReservedWordNumber + NonReservedWordNumber$$

Em contrapartida, na regra do não-terminal <*SQL special character*> da Figura 5.7 da gramática de SQL 92, podemos observar um exemplo de tipos de terminais que não são *keywords*, esses são considerados na gramática operadores, pontuações e caracteres especiais. Dando importância a esses tipos, elaboramos uma métrica para calcular o somatório desses terminais. Dessa maneira foi possível investigarmos a expressividade por outro ponto de vista, uma vez que essa adição pode mostrar que novas operações significativas foram incluídas na linguagem.

Comparando a regra de <*SQL special character*> das Figuras 5.7 e 5.8, observamos a adição dos terminais como <*circumflex*>, <*left brace*> e <*right brace*>, e além disso verificamos também o refatoramento na gramática de 99 relacionados aos terminais <*left bracket*> e <*right bracket*>, uma vez que esses terminais da regra de produção <*SQL embedded language character*> de 92, foram adicionados diretamente em <*SQL special character*>.

Então, definimos a **Métrica II – Número de terminais que não são *keywords*** para calcular a expressividade da linguagem considerando os terminais que não são palavras-chave da gramática. Para isso consideramos a regra <*SQL language character*>, ver a Figura 5.6, que engloba a regra <*SQL special character*> apresentada acima, além das regras <*simple Latin letter*> e <*digit*> que mostram as letras maiúsculas, minúsculas e dígitos que pertencem a linguagem.

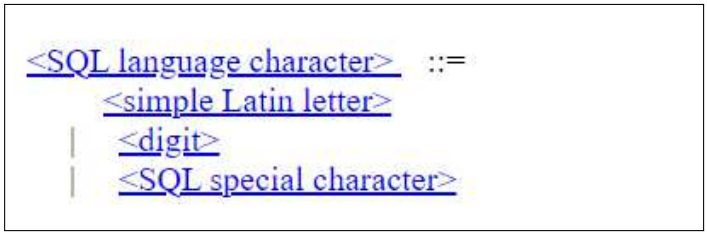
Dessa maneira, sendo *NonKeywordsTerminalNumber* o número total de terminais que não são *keywords*, a Métrica II é definida pelo somatório de *operatorsNumber* o número de operadores da gramática, *specialCharactersNumber* o número de caracteres especiais, *punctuationsNumber* o número de pontuações, *digitsNumber* o número de dígitos numéricos e *lettersNumber* o número de letras maiúsculas e minúsculas.

Métrica II – Número de terminais que não são *keywords*

$$NonKeywordsTerminalNumber = operatorsNumber + specialCharactersNumber + punctuationsNumber + digitsNumber + lettersNumber$$

O número de dígitos numéricos e de letras maiúsculas e minúsculas, por mais que não mudem de uma gramática para outra, são considerados como terminais que não *keywords* da linguagem e por isso foram incluídos na Métrica II.

Figura 5.6: Regras de produção <*SQL language character*>



Fonte: Gramáticas BNF – SQL 1992 [46] e 1999 [47]

Figura 5.7: Regra de produção <*SQL special character*> na gramática de 92

Fonte: Gramática BNF – SQL 1992 [46]

Figura 5.8: Regra de produção <*SQL special character*> na gramática de 99

Fonte: Gramática BNF – SQL 1999 [47]

Diante das métricas expostas até aqui, criamos a **Métrica III – Número total de terminais na linguagem** derivada das anteriores, que tem como objetivo calcular o somatório dos terminais da gramática. Sendo esses terminais conhecidos como *KeywordsTerminalNumber* e *NonKeywordsTerminalNumber*, onde equivalem respectivamente ao número de terminais que são do tipo *keywords* e o número terminais que não são do tipo *keywords*.

Métrica III – Número total de terminais na linguagem

$$TerminalNumber = KeywordsTerminalNumber + NonKeywordsTerminalNumber$$

A Métrica III mostrará uma visão geral dos terminais nas versões de SQL 92 e 99 das gramáticas, considerando os tipos existentes. Essa diferença dos terminais nas gramáticas indica a evolução na linguagem, bem como o ganho da expressividade.

Agora estendendo nossa análise incluindo os não-terminais das gramáticas de SQL, buscamos também compreender como a expressividade poderia ser medida. Observamos que quando uma regra de produção adiciona novos não-terminais, como exemplo do não-terminal criado <*large object length*> incluído nas regras de produção dos novos tipos de dados das Figuras 5.2, 5.3 e 5.5, verificamos que a regra de produção <*large object length*> adiciona também novos não-terminais para a gramática, como ilustrado na Figura 5.9.

Figura 5.9: Regra de produção <*large object length*> na gramática de 99

SQL - 99

<large object length> ::= <unsigned integer> [<multiplier>] | <large object length token>

<multiplier> ::= K | M | G

<large object length token> ::= <digit> ... <multiplier>

Fonte: Gramática BNF – SQL 1999 [47]

Esses novos não-terminais <*multiplier*> e <*large object length token*> criados também na gramática de 99, acrescentam a linguagem SQL novos conceitos que vieram acompanhados com a inclusão do não-terminal <*large object length*>. Dessa forma, o ganho de

expressividade da linguagem também pode ser medido considerando os novos não-terminais adicionados na gramática. Tendo em vista que esses novos não-terminais definem novas regras de produção que constam terminais ou não-terminais da gramática.

Então, definimos a **Métrica IV – Terminais por regras de produção** para calcularmos a expressividade da linguagem considerando os terminais e regras de produção da gramática, dado que o número de terminais criados na gramática está diretamente relacionado com o número de regras de produção existentes.

Dessa maneira, tem-se *TerminalsPerRule* como o número de terminais por regras de produção; *TerminalNumber* como sendo o número total de terminais; e *PRNumber* o número total de regras de produção da gramática. A Métrica IV é calculada pela divisão de *TerminalNumber* por *PRNumber*.

Métrica IV – Terminais por regras de produção

$$TerminalsPerRule = \frac{TerminalNumber}{PRNumber}$$

Por fim, consideramos para o cálculo de *PRNumber* a quantidade de não-terminais definidos no lado esquerdo, ou seja, na cabeça de uma regra de produção tanto na gramática de SQL 92 como também de SQL 99.

Capítulo 6

Avaliação

6.1 Resultados e Discussão

Nesta seção serão apresentados os resultados e discussões das métricas criadas para avaliação de expressividade da linguagem SQL por meio das gramáticas de 1992 e 1999.

6.1.1 Métrica I – Número de terminais que são *keywords*

Considerando as gramáticas em estudo, SQL 92 e SQL 99, a quantidade das palavras-chave foi calculada manualmente da Métrica I, onde verificamos o total de 276 e 433 *keywords* definidas pelo somatório de *reserved words* e *non-reserved words*. Com a diferença de 157 palavras no geral, sendo 69 palavras reservadas e 88 não reservadas, conforme descrito na Tabela 6.1.

	Reserved words	***Non-reserved words***
SQL 92	226	50
SQL 99	295	138

Tabela 6.1: Número das *keywords* em SQL

A partir do que observamos, a versão de SQL 99 apresenta suporte maior de *keywords* com relação à SQL 92, mas esse número de *reserved-words* e *non-reserved words* refletiu não apenas em novos conceitos para linguagem, e por isso visando realizar um melhor entendimento e detalhamento dessa evolução das *keywords*, as tabelas SQL *Reserved words* e

SQL *Non-reserved words* (veja nos Apêndices B e C) foram elaboradas para dar suporte às análises a seguir.

A Tabela SQL *Reserved words*, do Apêndice B, apresenta todas as palavras reservadas da linguagem em 92 e 99, como também uma pequena descrição para algumas delas. Observamos que dentre as 295 palavras de SQL 99, o total 205 continuaram as mesmas da versão anterior. O total de 69 novas *reserved words* inseridas estavam relacionadas a: tipos de dados, comandos condicionais, comandos iterativos, funções, classes, registros e exceções. Desse modo, na Tabela 6.2 mostramos a classificação por categorias, na qual consideramos as mudanças dessas palavras reservadas da versão de 99 em SQL. As quatro categorias foram classificadas como: tipos de dados, comandos, expressões e exceções, representando onde a linguagem obteve maiores ganhos em expressividade.

	Exemplos de terminais reservados adicionados em SQL 99
Tipos de dados	*ARRAY*, *BINARY*, *BLOB*, *BOOLEAN*, *CLOB*, *NCLOB*
Comandos	*IF*, *ELSEIF*, *WHILE*, *LOOP*
Expressões	*AFTER*, *BEFORE*, *CALL*, *CUBE*, *GROUPING*, *TRIGGER*, *CURRENT_DEFAULT_TRANSFORM_GROUP*, *CURRENT_PATH*, *CURRENT_ROLE*
Exceções	*SQLEXCEPTION*, *SQLWARNING*

Tabela 6.2: Classificação por categorias de terminais reservados

Na categoria de tipos de dados apresentamos que novos construtores como *ARRAY*, *BINARY*, *BLOB*, *BOOLEAN*, *CLOB* e *NCLOB* foram incluídos em 99. Essas adições promoveram um maior suporte quanto a novos tipos de dados para a linguagem, pois operações que antes eram mais complexas de serem escritas em SQL 92 sem esses tipos de dados tornaram-se mais fáceis de expressá-las na linguagem de SQL 99.

Estruturas de comandos como *IF*, *ELSEIF*, *WHILE* e *LOOP* passaram a existir também na versão de 99, representando alguns tipos de comandos condicionais e de iteração da linguagem. Os ganhos dessas estruturas para a linguagem são inúmeros, a partir do seu uso o usuário é capaz de realizar diferentes funções de forma prática, além de contribuir no controle do fluxo de execução das consultas em SQL, diminuir a complexidade, melhorar a

performance e expressividade, entre outros.

Quanto à categoria de expressões, notamos a adição de funções associadas a eventos especiais que podem ser utilizados em tabelas, sendo algumas dessas: *AFTER*, *BEFORE*, *CALL*, *CUBE*, *GROUPING* e *TRIGGER*. Outros, são alguns novos tipos de registros como *CURRENT_DEFAULT_TRANSFORM_GROUP*, *CURRENT_PATH* e *CURRENT_ROLE*.

Por último, na categoria de exceções, como exemplos citamos *SQLEXCEPTION* e *SQLWARNING* que foram adicionados em 99. As adições dessas *reserved words* para tratamento de erros na linguagem evidencia mais uma vez os ganhos na expressividade, uma vez que com o tratamento de exceções é indicado ao usuário uma forma mais amigável para resolver o problema, evitando por outro lado que a execução do código seja interrompida anormalmente e abruptamente.

Referente ainda às mudanças das palavras reservadas, percebemos também por meio das análises que 19 palavras deixaram de ser *reserved words* e passaram a ser *non-reserved words* em SQL 99, são alguns exemplos *UPPER*, *LOWER*, *MAX*, *MIN* e *SUM*. Essas *keywords* passaram a apresentar em SQL 99 algum significado especial apenas em contextos específicos, ou seja, quando são usadas como função de retornar algo para alguma expressão em SQL. Essa mudança de refatoramento para *non-reserved words* não afetou na função de nenhuma delas na linguagem.

Por fim, verificamos também que 2 palavras reservadas *SQLCODE* e *SQLERROR* foram excluídas da gramática, onde *SQLERROR* foi substituída pelas novas *reserved words* para tratamento de exceções *SQLEXCEPTION* e *SQLWARNING*. Enquanto, *SQLCODE* foi excluída devido as mudanças ocorridas nas regras de produção da seção de *Procedures* da gramática de SQL 99.

É importante notar que as novas operações permitidas na linguagem de SQL 99 buscaram melhorar em termos de capacidade de escrita para os novos usuários, uma vez que por meio da adição das palavras-chave nas categorias tipos de dados, comandos, expressões, exceções das palavras reservadas facilitou em expressar código de forma mais rápida, clara, curta e concisa, aspectos esses observados e levados em consideração quando nos referimos à expressividade de uma linguagem. Alguns exemplos de códigos escritos em SQL 92 e 99 mostrando essas mudanças foram apontados na Seção 2.4 do presente estudo.

Agora com relação à Tabela SQL *Non-reserved words*, do Apêndice C, que mostra todas

as palavras não-reservadas, visualizamos que dentre o total de 138 palavras de SQL 99, 48 continuaram iguais quando comparadas com as palavras da gramática de SQL 92. Além disso, observamos que 19 palavras reservadas que faziam parte de 92, passaram a ser *non-reserved words* na gramática de 99, como reportado nos dois últimos parágrafos.

Por outro lado, 71 novas *non-reserved words* adicionadas estavam relacionadas a funções, sendo algumas delas *COUNT*, *MOD*, *OVERLAY*, *ABS* e *POSITION*. Outras foram criadas como cláusulas *INVOKER*, *INSTANTIABLE*, *IMPLEMENTATION*, *SCOPE* e *STYLE*. Também algumas palavras não-reservadas foram adicionadas para especificar métodos como *INSTANCE* e *SELF*. Predicados como *CONTAINS*. E por último, outras palavras são complementos para *reserved words*, criadas para realizar eventos especiais, como exemplos *TRIGGER_CATALOG*, *TRIGGER_SCHEMA* e *TRIGGER_NAME*.

Verificamos ainda que 1 palavra não-reservada *DYNAMIC_FUNCTION* foi excluída e 1 palavra *DATA* passou a ser *reserved word* em 99, ao invés de *non-reserved*. Essa mudança de refatoramento da palavra-chave *DATA*, não afetou na função dela na linguagem.

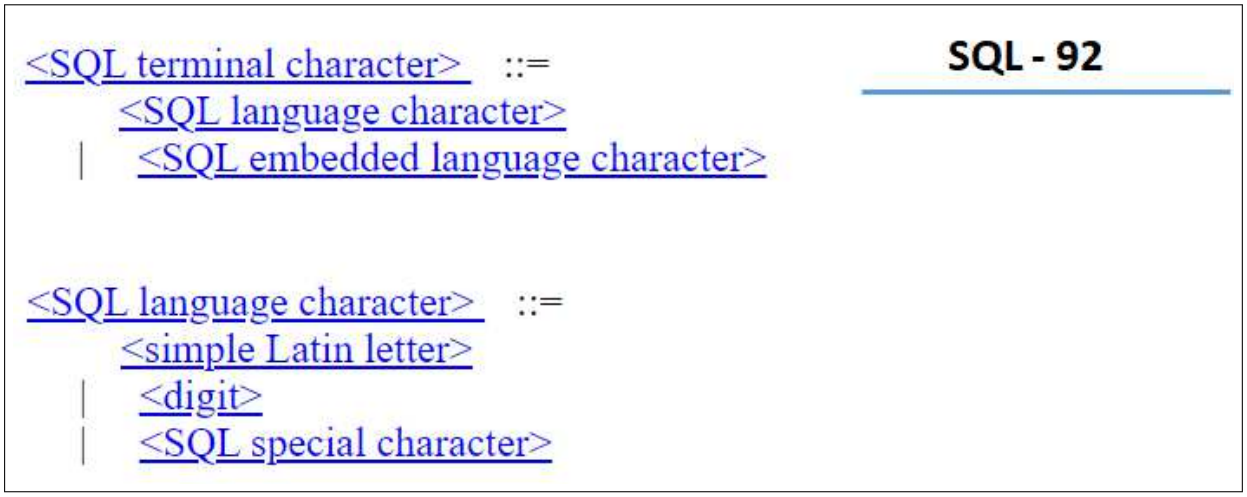
As *non-reserved words* adicionadas na linguagem em SQL 99, mostram sua importância na expressividade tão quanto as *reserved words*. Tendo em vista que elas terminam sendo utilitárias para o usuário, uma vez que termina ajudando na escrita dos seus códigos em diferentes contextos, reduzindo linhas de códigos que facilmente poderiam serem maiores expressas se caso elas não existissem.

6.1.2 Métrica II – Número de terminais que não são *keywords*

Na gramática SQL 92, investigamos que o não-terminal <*SQL terminal character*> define quais seriam os outros tipos de terminais presentes na linguagem, veja Figura 6.1. Eles são apresentados nas regras <*SQL language character*> e <*SQL embedded language character*>, e melhores detalhados nas tabelas SQL *Terminals* e SQL *Special Characters*, veja nos Apêndices D e E.

Ao aplicar Métrica II, o cálculo do somatório obteve como total 85 terminais que não são *keywords*, onde 52 são letras, maiúsculas e minúsculas, 10 dígitos, 21 caracteres especiais e 2 *SQL embedded language character* (colchete direito e esquerdo).

Figura 6.1: Regras de produção 92 <*SQL terminal character*> e <*SQL language character*>



Fonte: Gramática BNF – SQL 1992 [46]

Por outro lado, na gramática de 99 (ver Figura 6.2), a regra de <*SQL terminal character*> mostra uma pequena diferença em que o não-terminal <*SQL embedded language character*> foi excluído, e os colchetes (direito e esquerdo) que eram definidos nesta regra passaram a serem vistos diretamente na de <*SQL special character*> (rever Apêndices D e E).

Figura 6.2: Regras de produção 99 <*SQL terminal character*> e <*SQL language character*>

Fonte: Gramática BNF – SQL 1999 [47]

O total de 88 terminais foram identificados quando utilizamos a Métrica II na gramática de 99. O cálculo somou as 52 letras, 10 dígitos e 26 caracteres especiais. Entre os caracteres especiais adicionados, foram esses: circunflexo (^) e a chave esquerda ({) e direita (}).

Com relação à adição do caractere especial circunflexo, a regra <*regular character set*> vista na Figura 6.3 foi definida na gramática de 99 como um novo suporte para definir ex-

pressões regulares, em específico no uso com predicado *SIMILAR TO* também adicionado como *reserved word* na linguagem durante o mesmo ano.

O *SIMILAR TO* faz a correspondências de uma expressão de *string*, tal como um nome de coluna, com um padrão de expressão regular SQL. O operador *SIMILAR TO* funciona de forma semelhante do predicado *LIKE* já existente na linguagem SQL 92, que tem a função de comparar uma expressão de cadeia de caracteres a um padrão em uma expressão SQL.

Um exemplo da diferença dos operadores *LIKE* e *SIMILAR TO*, pode ser visto abaixo, uma vez que para combinar apenas consoantes com *LIKE* é necessário declarar da seguinte maneira:

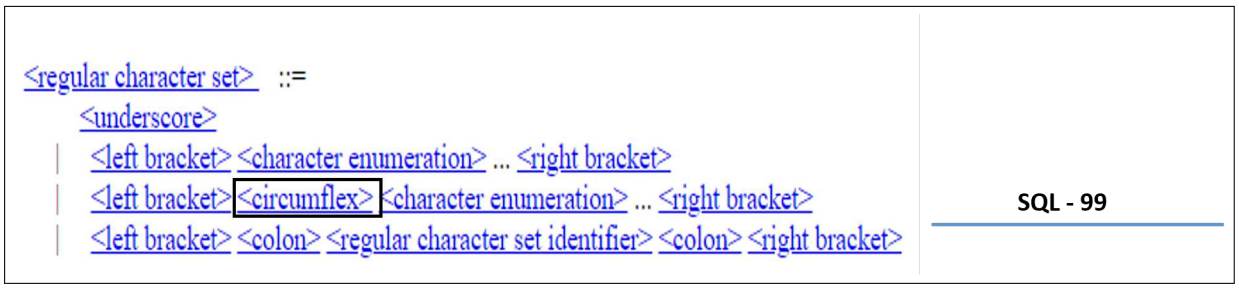
$$[BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz]$$

Enquanto que com *SIMILAR TO*, pode ser reduzido a:

[A-Za-z^AEIOUaeiou]

Dado que, quando aplicamos o circunflexo entre $A-Za-z$ e as vogais maiúsculas e minúscula, vimos que a forma escrita com expressão regular ajudou na concisão da linguagem, expressando o necessário com brevidade e clareza.

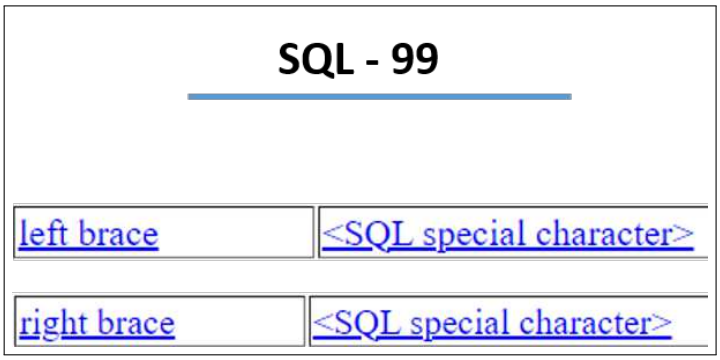
Figura 6.3: Regra de produção 99 <*regular character set*>



Fonte: Gramática BNF – SQL 1999 [47]

Por outro lado, as chaves adicionadas direita e esquerda em SQL 99, observamos na gramática que não contribuiu expressivamente em termos de criação de novas regras de produção. Na Figura 6.4, mostra a tabela de referência cruzada das regras de produção associadas a <*left brace*> e <*right brace*> mostrando que apenas pertencem a <*SQL special character*>.

Figura 6.4: Cross-Reference Table: <*left brace*> e <*right brace*> rules



Fonte: Gramática BNF – SQL 1999 [47]

6.1.3 Métrica III – Número total de terminais na linguagem

Levando em consideração as análises dos resultados feitas das Métrica I – Número de terminais que são *keywords* e Métrica II – Número de terminais que não são *keywords*, a Métrica III é derivada dessas anteriores. Como resultado, a Tabela 6.3 mostra os valores para cada um dos dois tipos de terminais, onde obteve ao total 361 e 521 em SQL 92 e SQL 99, respectivamente.

	Keywords	***Non-keywords***
SQL 92	276	85
SQL 99	433	88

Tabela 6.3: Número total de terminais na linguagem

Esse aumento expressivo de 160 novos terminais considerando as duas gramáticas em estudo, nos levou a refletir sobre a adição de novas regras de produção, refatoramento de regras de produção já existentes, bem como a relevância, de forma geral, desses terminais e não-terminais para a linguagem. Para isso, a Métrica IV – Terminais por regras de produção da Subseção 6.1.4 seguinte, estenderá e mostrará resultados aplicando agora as regras de produção das gramáticas SQL 92 e SQL 99.

6.1.4 Métrica IV – Terminais por regras de produção

Calculamos a Métrica IV através do número de terminais exposto na Métrica III pelo número de regras de produção existentes na tabela *Cross-Reference Table: Rules* definidas nas gramáticas de SQL 92 [46] e SQL 99 [47].

Inicialmente, aplicamos a Métrica IV na gramática de SQL 92, onde obtivemos o valor de 0,5730 com a divisão dos 361 terminais por 630 regras de produção. Observe o cálculo representado logo abaixo.

$$TerminalsPerRule = \frac{361}{630} = 0,5730$$

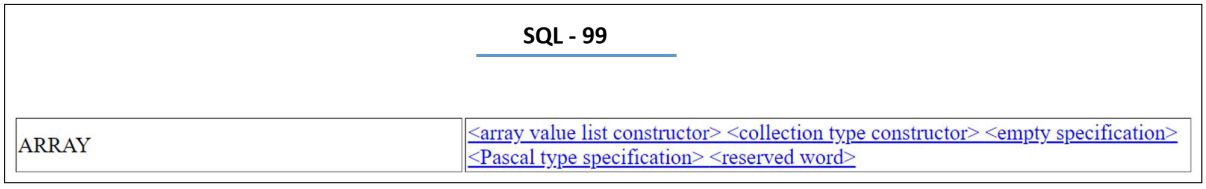
Já na gramática de SQL 99, ao calcularmos a divisão de 521 terminais pelo número de 1244 regras de produção existentes, obtivemos o valor de 0,4188, como ilustrado a seguir.

$$TerminalsPerRule = \frac{521}{1244} = 0,4188$$

Constatamos com esses resultados, que o número de regras de produção da gramática de SQL 99 é quase o dobro de SQL 92, com a diferença de 614 regras. Além disso, observamos também que apesar do aumento de terminais e regras de produção na linguagem SQL 99, é visto que houve uma baixa do número de terminais por regras, sendo 0,4188.

Refletimos com relação esses resultados mostrados acima com a palavra-chave reservada *ARRAY* que foi adicionada na linguagem SQL em 99. Dessa forma, a partir da tabela de *Cross-Reference Table: Keywords* da versão de 99, verificamos os apontamentos das regras de produção associadas ao terminal *ARRAY*, ilustrada na Figura 6.5.

Figura 6.5: Cross-Reference Table: ARRAY *keyword*



SQL - 99

ARRAY	<array value list constructor> <collection type constructor> <empty specification> <Pascal type specification> <reserved word>

Fonte: Gramática BNF – SQL 1999 [47]

Com a adição do novo terminal *ARRAY* na linguagem SQL em 99, percebemos também a criação de 4 novos não-terminais <*array value list constructor*>, <*collection type constructor*>, <*empty specification*> e <*Pascal type specification*> na gramática. Logo, vale destacar

que esses novos não-terminais também acrescentam novas evoluções para a linguagem, ou seja, expressam também seus terminais ou não-terminais para a gramática.

Sendo assim, a Métrica IV apontou um valor alto (0,5730) de terminais pelo número de regras de produção da gramática de SQL 92, mostrando que nessa gramática mais da metade das regras de produção incluem esses terminais.

Por outro lado, mesmo que a Métrica IV tenha apresentado um valor baixo (0,4188) de terminais pelo número de regras de produção da gramática de SQL 99, devemos lembrar que essas mesmas regras estão diretamente relacionadas a esses novos terminais adicionados, uma vez que refatoramentos na gramática foram feitos e novos conceitos também foram introduzidos na linguagem.

Em vista disso, concluímos que a inclusão de mais não-terminais na gramática está também relacionada a expansão dos novos conceitos para na linguagem, impactando igualmente no ganho de expressividade da linguagem SQL tal como os terminais.

6.2 Implicações

Tratando agora sobre as implicações que este estudo traz para área de DSLs, observamos que a abordagem de avaliação usando GLCs que foi apresentada aqui não tem relação com nenhum outro trabalho que já se encontra desenvolvido na área. Muitas avaliações com DSLs são direcionadas unicamente para as linguagens que estão sendo investigadas e voltadas para medir características como complexidade e usabilidade, diferentemente deste trabalho que foca em uma abordagem que busca avaliar a característica de expressividade para qualquer tipo de DSL.

Verificamos ainda por meio dos resultados deste trabalho que as análises e discussões das métricas que foram mostradas revelam indícios concretos do aumento de expressividade da linguagem SQL estudada, por meio dos terminais e não-terminais das gramática. Outros trabalhos não exploram essas discussões de forma efetiva e se baseiam muitas das vezes em medir pela quantidade de linhas de código (LOC - *Lines Of Code*), ou através de formulários e questionários avaliativos aplicados com usuários da linguagem.

Logo, este trabalho traz uma iniciativa inovadora de avaliação de linguagens específicas utilizando GLCs para medir a característica de expressividade, onde aspiramos que outros

novos caminhos na área sejam executados tendo como base a abordagem que aqui foi demonstrada.

6.3 Ameaças à Validade

Esta seção tem como propósito a discussão das ameaças à validade do estudo seguindo a classificação proposta por Wohlin *et al.* [66] e as estratégias usadas para mitigá-las.

Validade de conclusão: a quantidade de *keywords*, não *keywords* e regras de produção foram calculadas de forma manual para realizarmos uma melhor análise das diferenças entre as versões das gramáticas de SQL 92 e 99. Entretanto, para mitigar essa ameaça foi realizada a checagem por meio do uso da ferramenta Excel [1], onde todas *keywords*, não *keywords* e regras de produção foram transferidas para ferramenta, visando confirmar os valores calculados pela ferramenta com os valores calculados manualmente.

Validade interna: o uso de gramáticas antigas da linguagem SQL dos anos de 1992 e 1999 para realizar o estudo de caso. Verificamos gramáticas de outros domínios, mas não estavam completas. Portanto, preferimos por manter as gramáticas de SQL 92 e 99 pela razão de ser uma linguagem bastante utilizada no mercado, e as gramáticas encontradas apresentam tamanho e complexidade satisfatória para os estudos realizados.

Validade de constructo: a construção das quatro métricas apresentadas neste estudo podem não terem sido suficientes para capturar em totalidade a expressividade de linguagens específicas de domínio. Assim, buscou-se garantir que essas métricas criadas, considerando o estudo de caso com as gramáticas de SQL, envolvessem os terminais como também os não-terminais das gramáticas, sendo esses dois grupos importantes para a identificação da evolução da linguagem.

Validade externa: o uso apenas das gramáticas de SQL 92 e 99 na aplicação das métricas de expressividade criadas, apresentam limitações quanto a generalização dos resultados para outras linguagens específicas de domínio. Portanto, este estudo deve ser replicado considerando o uso de outras gramáticas de DSLs para confirmar se as métricas contribuem na avaliação de expressividade dessas linguagens e, consequentemente, abordar questões de validade externa.

[1] https://www.microsoft.com/microsoft-365/excel

Capítulo 7

Trabalhos Relacionados

Visando compreender como as linguagens estão sendo avaliadas quanto à expressividade uma busca na literatura foi efetuada levando em consideração trabalhos que envolvessem tanto GPLs como DSLs, uma vez que existe a carência de estudos que investiguem sobre esta área em ambos tipos de linguagens. Desse modo, o trabalho de Miglioranza [34] teve como objetivo realizar um estudo comparativo das linguagens de programação Java e C++ para manipulação de vídeo em telefones móveis, em vista disso o critério de expressividade foi um dos pontos medidos. Os autores utilizam a abordagem de medir pela quantidade de linhas de código (LOC - *Lines Of Code*) escritas nas linguagens Java e C++ a função de iniciar a captura de imagens da câmera. Concluíram que Java apresentava maior expressividade (por ter menor quantidade de linhas escritas na função) e C++ menor expressividade, contudo maior flexibilidade porque permite o uso de mais recursos de configuração. Outro estudo de Berkholz [8] segue essa mesma linha para medição de expressividade em linguagens de programação. Ele faz análises em um grande conjunto de dados mostrando gráficos da distribuição de linhas de código por *commit* a cada mês por cerca de 20 anos, ponderada pelo número de *commits* em um determinado mês, feitos em linguagens de programação. Por fim, o autor mostra como as tendências fazem sentido e como a expressividade varia amplamente entre as linguagens considerando apenas a métrica LOC.

O trabalho de Kahraman e Bilgen [26] apresenta uma proposta de Framework para Avaliação Qualitativa de DSLs (FAQD) que pode ser utilizada no início ou fim do processo de desenvolvimento da DSL. Uma das características medidas pelo FAQD é a expressividade da DSL, bem como as subcaracterísticas relacionadas à expressividade. Para fazer essa

avaliação da DSL, os autores criaram dois formulários, um para o avaliador classificar a importância das características e outro para avaliação de sentenças das subcaracterísticas. Após a aplicação desses dois formulários, os resultados são gerados de forma automática por um último formulário o qual demonstra em que nível de sucesso encontra-se a DSL avaliada, podendo estar a DSL incompleta, satisfatória ou efetiva.

No trabalho de Haugen e Mohagheghi [22] é apresentada uma estrutura de questionário baseado em três dimensões de DSL — expressividade, transparência e formalização. Já o estudo de Hermans, Pinzger e Deursen [23] mostra fatores de sucesso de DSL, onde dentre um deles está a expressividade. Os autores fazem um estudo empírico usando o questionário proposto e avaliam os fatores de sucesso com um estudo de caso.

Por último, no trabalho de García-Magariño, Gómez-Sanz e Fuentes-Fernández [16] é proposto uma avaliação por meio de métricas quantitativas elaboradas para mensurar a qualidade de expressividade entre versões de uma linguagem de modelagem. Enquanto no artigo de Banerjee e Sarkar [5] um conjunto de métricas quantitativas foi apresentado como parte do framework de avaliação de qualidade para softwares baseados em componentes. Dentre as métricas criadas, uma delas está relacionada à expressividade, em que busca medir sob o ponto de vista do usuário (verificando como o sistema baseado em componentes é acessível aos usuários).

Já com relação a comparação dos estudos acima apresentados com o que foi realizado nesta dissertação de mestrado, observamos que nenhum estudo acima retrata uma abordagem de avaliação quantitativa por meio da criação de métricas para mensurar a característica de expressividade para qualquer tipo de DSL. Então, buscamos ajudar a área de estudo de avaliações de linguagens específicas de domínio através do uso das gramáticas, uma vez que elas são comuns para toda DSL e já se encontram desenvolvidas e são capazes de mensurar o aumento de expressividade considerando duas ou mais versões da linguagem desenvolvida.

Capítulo 8

Conclusões

Neste trabalho, apresentamos uma abordagem de avaliação de linguagens específicas de domínio por meio da criação de métricas de *software*, que foram baseadas em gramáticas, e são usadas para medir os ganhos de expressividade da linguagem SQL. Analisamos e definimos um conjunto de quatro métricas, onde três delas medem a estrutura dos terminais da gramática, enquanto a outra métrica mede a relação dos terminais e não-terminais da gramática.

Descrevemos como utilizar essas gramáticas livres de contexto de uma linguagem para obter os resultados quantitativos das métricas criadas. Apresentamos os resultados das nossas métricas comparando duas versões de gramáticas da linguagem SQL, comumente utilizada no mercado para lidar com banco de dados relacionais.

Com análise das métricas identificamos que a linguagem de SQL 99 apresentou mais ganhos na expressividade quando comparada com a linguagem SQL 92. Verificamos que o número de 160 terminais adicionados na gramática de SQL 99 teve relação direta para esta conclusão, uma vez que novos terminais reservados que foram adicionados em categorias como Tipos de dados, Comandos, Expressões e Exceções possuíam forte poder na melhoria de capacidade de escrita da linguagem.

Os resultados encontrados evidenciaram também a importância dos não-terminais para SQL 99 através da Métrica IV, uma vez que com adição de terminais, novos não-terminais também são expandidos para acompanhar esses novos conceitos que foram incorporados a linguagem, e consequentemente impactam no aumento de expressividade.

Mostramos também neste trabalho, especificamente na Seção 2.3, uma classificação simplificada de dois tipos de mudanças encontradas em gramáticas livres de contexto, sendo elas

chamadas de *mudanças diretas* e *mudanças de aperfeiçoamentos*. Essas investigações das mudanças nas gramáticas SQL 92 e 99 abriram os caminhos para as definições das métricas de expressividade que aqui são expostas.

Por fim, é importante reforçar que as métricas propostas neste trabalho são focadas nos artefatos como regras de produção, terminais e não-terminais das gramáticas livres de contexto de linguagens específicas de domínio e árvores de derivação, que são principalmente utilizados durante a etapa de análise sintática de um compilador. Outros artefatos, de outras etapas de compiladores, a exemplo da análise léxica ou da análise semântica, não foram considerados no trabalho.

8.1 Contribuições

Em resumo, as principais contribuições deste estudo foram:

1. Ajudar a área de estudo de avaliações de linguagens específicas de domínio através do uso das gramáticas, que já se encontram desenvolvidas, para medir a característica de qualidade de expressividade.

2. A análise dos tipos de mudanças encontradas, considerando as duas versões das gramáticas de SQL. Tendo em vista que essa análise permitiu uma base de entendimento para propositura das métricas.

3. A criação de quatro métricas para apoiar na análise quantitativa de DSLs. Tais métricas buscam capturar os ganhos de expressividade da linguagem específica de domínio por meio dos terminais e não-terminais da gramática.

4. A avaliação das métricas propostas através do estudo de caso com as gramáticas de 1992 e 1999 da linguagem SQL, onde através das análises dos resultados das métricas evidenciamos os ganhos de expressividade maior da linguagem SQL 99.

8.2 Limitações e Trabalhos Futuros

Considerando que o escopo apresentado neste trabalho resultou em algumas limitações, é ciente que a abordagem proposta pode ser aprimorada para alcançar melhores resultados.

Sendo assim, as principais limitações estão elencadas a seguir:

- Os resultados alcançados foram obtidos por meio da avaliação de apenas duas gramáticas da linguagem SQL. Não realizamos um estudo mais detalhado considerando gramáticas de outros domínios.
- Apenas uma única métrica foi criada envolvendo os não-terminais da gramática, consideramos que outras métricas incluindo os não-terminais podem ser estendidas através de novas análises.
- A análise da classificação por categorias de terminais apresentadas ao avaliar a Métrica I foi bastante promissora. Consideramos que novas métricas de expressividade poderiam ter sido exploradas levando em conta as categorias de tipos de dados, comandos, expressões e exceções.

Logo, os resultados obtidos nesta dissertação propiciam que trabalhos futuros sejam realizados para auxiliar na generalização das métricas criadas para um maior número de linguagens específicas de domínio. Além disso, a análise da classificação por categorias de terminais apresentadas na Subseção 6.1.1 pode ser aprofundada por meio da criação de novas métricas. E por fim, uma melhor análise também dos não-terminais pode ser realizada com a definição de novas métricas.

Como trabalhos futuros, pretende-se ampliar o conjunto de métricas de expressividade elaboradas, definindo outras que considerem a classificação por categorias de terminais, como também outras métricas para os não-terminais da gramática. Pensa-se também como trabalhos futuros, definir um *framework* para retornar os resultados quantitativos do conjunto de métricas propostas pela autora. Espera-se ainda que outros pesquisadores interessados na área de DSLs e linguagens repliquem o estudo com outras linguagens específicas de domínio, o que poderia melhorar a confiança das métricas desenvolvidas.

Bibliografia

[1] *JFlex. Disponível em: <https://gnuwin32.sourceforge.net/packages/flex.htm>.*

[2] Significado de expressiva. *Priberam Dicionário*, 2021.

[3] Significado de expressividade. *Dicio*, 2021.

[4] Diego Albuquerque, Bruno Cafeo, Alessandro Garcia, Simone Barbosa, Silvia Abrahão, and António Ribeiro. Quantifying usability of domain-specific languages: An empirical study on software maintenance. *Journal of Systems and Software*, 101:245 – 259, 2015.

[5] Prasenjit Banerjee and Anirban Sarkar. Quality evaluation framework for component based software. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, pages 1–6, 2016.

[6] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. Quality in use of domain-specific languages: A case study. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU '11, pages 65–72, New York, NY, USA, 2011. ACM.

[7] Brigham Bell, Wayne V Citrin, Clayton Lewis, and John Rieman. The programming walkthrough: A structured method for assessing the writability of programming languages; cu-cs-577-92. *Computer Science Technical Reports, vol. Paper*, 554, 1992.

[8] Donnie Berkholz. Programming languages ranked by expressiveness, March 2013.

[9] Charles Consel, Fabien Latry, Laurent Réveillere, and Pierre Cointe. A generative programming approach to developing dsl compilers. In *International Conference on Generative Programming and Component Engineering*, pages 29–46. Springer, 2005.

[10] Riley Evans, Samantha Frohlich, and Meng Wang. Circuitflow: A domain specific language for dataflow programming. In *Practical Aspects of Declarative Languages: 24th International Symposium, PADL 2022, Philadelphia, PA, USA, January 17–18, 2022, Proceedings*, pages 79–98. Springer, 2022.

[11] Microsoft Excel. *Excel. Disponível em: <https://www.microsoft.com/microsoft-365/excel>*.

[12] NE Fenton and SL Pfleeger. Software metrics: A rigorous and practical approach: Brooks, 1998.

[13] Martin Fowler. Language workbenches: The killer-app for domain specific languages, 2005.

[14] Martin FOWLER. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[15] Pedro Gabriel, Miguel Goulão, and Vasco Amaral. Do software languages engineers evaluate their languages? *CoRR*, abs/1109.6794, 2011.

[16] Iván García-Magariño, Jorge J Gómez-Sanz, and Rubén Fuentes-Fernández. An evaluation framework for mas modeling languages based on metamodel metrics. In *International Workshop on Agent-Oriented Software Engineering*, pages 101–115. Springer, 2008.

[17] Antonio Carlos Gil. *Métodos e técnicas de pesquisa social*. 6. ed. Ediitora Atlas SA, 2008.

[18] Graphviz. *DOT Language. Disponível em: <https://graphviz.org/doc/info/lang.html>*.

[19] A. GURGEL. Blending and reusing rules for architectural degradation prevention, pucrio, 2012.

[20] GURU99. *Gherkin Language: Format, Syntax Gherkin Test in Cucumber. Disponível em: <https://www.guru99.com/gherkin-test-cucumber.html>*.

[21] Ivan Halupka and Ján Kollár. Catalog of grammar refactoring patterns. *Open Computer Science*, 4(4):231–241, 2014.

[22] Oystein Haugen and Parastoo Mohagheghi. A multi-dimensional framework for characterizing domain specific languages. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07), Computer Science and Information System Reports*, 2007.

[23] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Domain-specific languages in practice: A user study on the success factors. In *International Conference on Model Driven Engineering Languages and Systems*, pages 423–437. Springer, 2009.

[24] Bernhard G. HUMM and Ralf S. ENGELSCHALL. Language-oriented programming via DSL stacking. In José A. Moinhos Cordeiro, Maria Virvou, and Boris Shishkov, editors, *ICSOFT 2010 - Proceedings of the Fifth International Conference on Software and Data Technologies, Volume 2, Athens, Greece, July 22-24, 2010*, pages 279–287. SciTePress, 2010.

[25] IBM. Palavras e palavras-chave reservadas sql.

[26] Gökhan Kahraman and Semih Bilgen. A framework for qualitative assessment of domain-specific languages. *Software & Systems Modeling*, 14(4):1505–1526, Oct 2015.

[27] TOLVANEN J.P. KELLY, S. Domain-specific modeling enabling full code generation. wiley, new york, 2008.

[28] Donald E Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736, 1964.

[29] Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77 – 91, 2016.

[30] LANGLOIS, Consuela elena Jitia, and Eric Jouenne. Dsl classification, 2007.

[31] Kenneth C Louden and Kenneth A Lambert. *Programming languages: principles and practices*. Cengage Learning, 2011.

[32] Paulo Blauth Menezes. *Linguagens formais e autômatos*. Sagra-Dcluzzato, 2005.

[33] Microsoft. Identificadores de banco de dados.

[34] Ewerton Felipe Miglioranza. Estudo comparativa de linguagens de programacao para manipulacao de video em telefones móveis. 2009.

[35] Naouel MOHA and Yann-Gael GUÉHÉNEUC. Decor: A tool for the detection of design defects. In *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, page 527–528, New York, NY, USA, 2007. Association for Computing Machinery.

[36] Thijs Molendijk. Dynamix: A domain-specific language for dynamic semantics. 2022.

[37] Clint MORGAN, Kris De Volder, and Eric Wohlstadter. A static aspect language for checking design rules. In *Proceedings of the 6th International Conference on Aspect-Oriented Software Development*, AOSD '07, page 63–72, New York, NY, USA, 2007. Association for Computing Machinery.

[38] Mozilla. *HTML: Linguagem de Marcação de Hipertexto. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>*.

[39] Nuno Oliveira, Maria João Pereira, Pedro Rangel Henriques, and Daniela Cruz. Domain specific languages: A theoretical survey. *INForum'09-Simpósio de Informática*, 2009.

[40] Michael Quinn Patton. *Qualitative research & evaluation methods*. sage, 2002.

[41] Ildevana Poltronieri, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. Usa-dsl: Usability evaluation framework for domain-specific languages. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 2013–2021, New York, NY, USA, 2018. Association for Computing Machinery.

[42] Ildevana Poltronieri, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. Usability evaluation framework for domain-specific language: A focus group study. *SIGAPP Appl. Comput. Rev.*, 18(3):5–18, October 2018.

[43] James F Power and Brian A Malloy. A metrics suite for grammar-based software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):405–426, 2004.

[44] S. G. D. PRADO. Linguagens livres de contexto, 2019.

[45] Leila Samimi-Dehkordi, Alireza Khalilian, and Bahman Zamani. Applying programming language evaluation criteria for model transformation languages. *International Journal of Software & Informatics*, 10(4), 2016.

[46] Ron Savage. Bnf grammar for iso/iec 9075:1992 - database language sql (sql-92).

[47] Ron Savage. Bnf grammar for iso/iec 9075:1999 - database language sql (sql-99).

[48] C. SCHMITT, S. Kuckuk, H. Köstler, F. Hannig, and J. Teich. An evaluation of domain-specific language technologies for code generation. In *2014 14th International Conference on Computational Science and Its Applications*, pages 18–26, June 2014.

[49] Robert W Sebesta. *Conceitos de Linguagens de Programação-11*. Bookman Editora, 2018.

[50] Stefan Sobernig, Patrick Gaubatz, Mark Strembeck, and Uwe Zdun. Comparing complexity of api designs: An exploratory experiment on dsl-based framework integration. *ACM SIGPLAN Notices*, 47(3):157–166, 2011.

[51] Software.informer. *Yacc. Disponível em: <https://yacc.software.informer.com/0.4/>*.

[52] SourceForge. *Bison. Disponível em: <https://gnuwin32.sourceforge.net/packages/bison.htm>*.

[53] SourceForge. *Flex. Disponível em: <https://gnuwin32.sourceforge.net/packages/flex.htm>*.

[54] SourceForge. *xText. Disponível em: <https://www.eclipse.org/Xtext/>*.

[55] Mark STREMBECK and Uwe ZDUN. An approach for the systematic development of domain-specific languages. *Software: Practice and Experience*, 39(15):1253–1292, 2009.

[56] Ricardo TERRA and Marco Tulio VALENTE. A dependency constraint language to manage object-oriented software architectures. *Softw. Pract. Exper.*, 39(12):1073–1094, August 2009.

[57] Naoyasu UBAYASHI, Jun Nomura, and Tetsuo Tamai. Archface: A contract place where architectural design and code meet together. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, page 75–84, New York, NY, USA, 2010. Association for Computing Machinery.

[58] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and GH Wachsmuth. Dsl engineering-designing, implementing and using domain-specific languages. 2013.

[59] Wikipédia. *LaTeX. Disponível em: <https://en.wikipedia.org/wiki/LaTeX>.*

[60] Wikipédia. *Poder expressivo (ciência da computação). Disponível em: <https://en.wikipedia.org/wiki/Expressive*$_p ower_(computer_s cience) >$.

[61] Wikipédia. *SQL. Disponível em: <https://pt.wikipedia.org/wiki/SQL>.*

[62] Wikipédia. *Verilog. Disponível em: <https://pt.wikipedia.org/wiki/Verilog>.*

[63] Wikipédia. *VHDL. Disponível em: <https://pt.wikipedia.org/wiki/VHDL>.*

[64] Wikipédia. *Wolfram Language. Disponível em: <https://en.wikipedia.org/wiki/Wolfram_Mathematica>.*

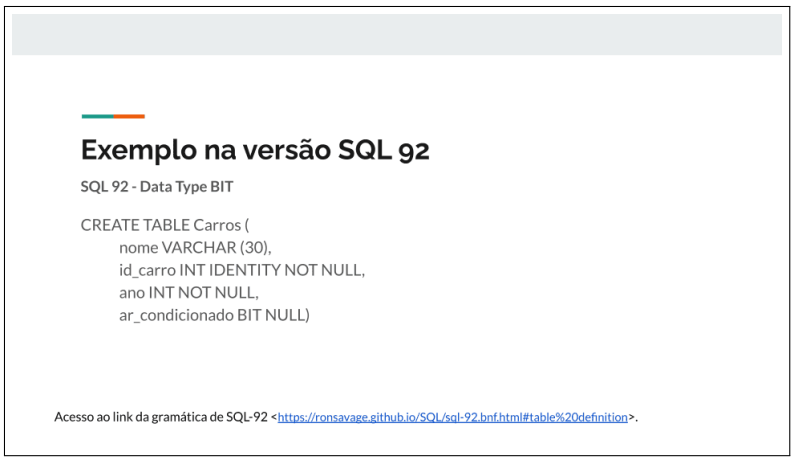
[65] Wikipédia. *XML. Disponível em: <https://pt.wikipedia.org/wiki/XML>.*

[66] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

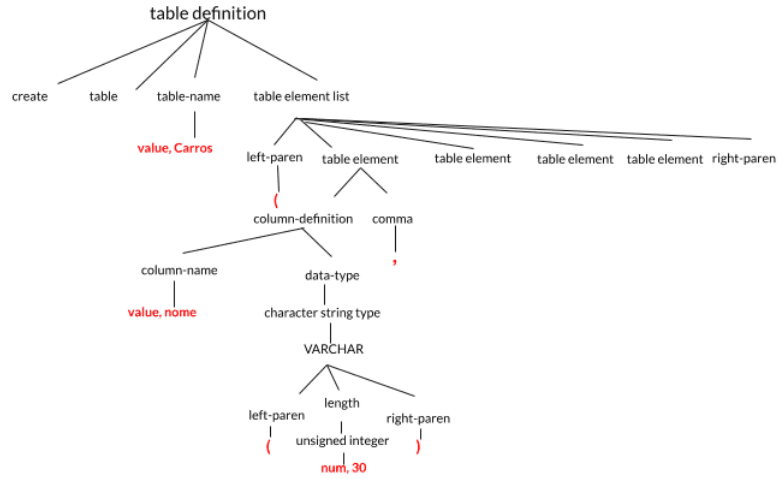
Apêndice A

Árvores de Derivação

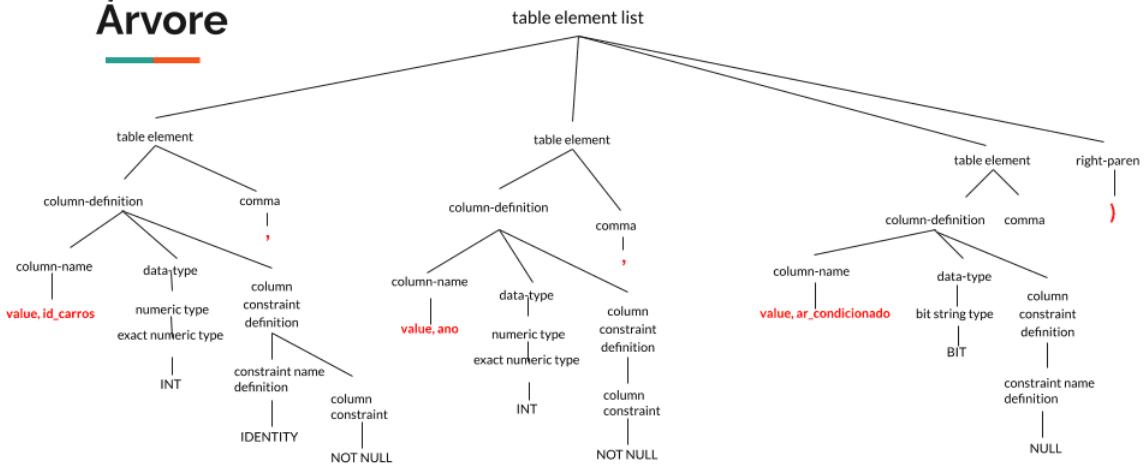
- **Construção do exemplo em SQL 92 com create table**



Árvore

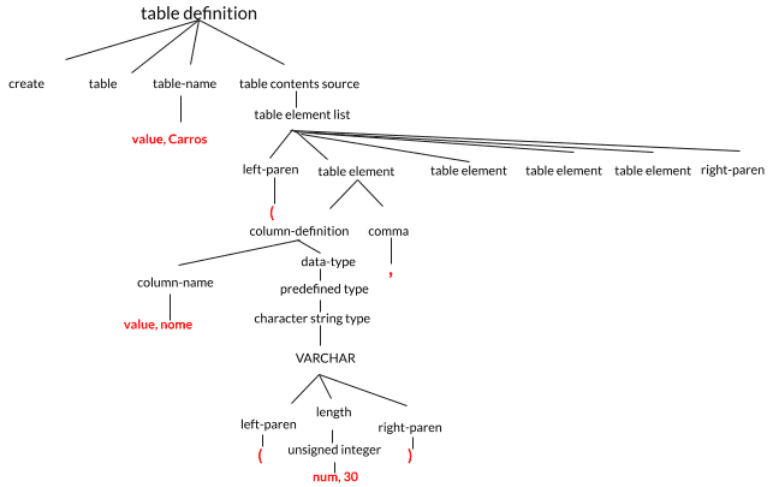


Árvore

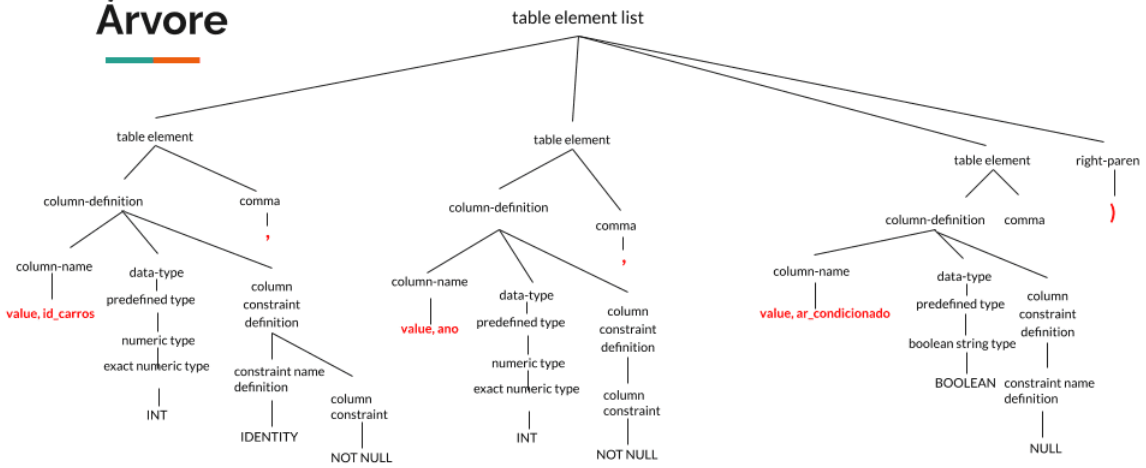


- **Construção do exemplo em SQL 99 com create table**

Árvore



Árvore



- **Análise das mudanças entre as gramáticas de 92 e 99**

	SQL - 92	SQL - 99	Reflexo
criação de tabela	<table definition> ::= CREATE [{ GLOBAL | LOCAL} TEMPORARY] TABLE <table name> <table element list> [ON COMMIT { DELETE | PRESERVE } ROWS]	<table definition> ::= CREATE [**<table scope>**] TABLE <table name> **<table contents source>** [ON COMMIT **<table commit action>** ROWS]	- Adição de 3 novas regras de produção **<table scope>** ::= <global or local> TEMPORARY **<table contents source>** ::= <table element list> | OF <user-defined type> [<subtable clause>] [<table element list>] **<table commit action>** ::= PRESERVE | DELETE - O não terminal **<table scope>** gera uma nova regra de produção contendo o não terminal *<global or local>* e o terminal *(TEMPORARY)*; - *<global or local>* gera uma nova regra de produção com dois terminais *(GLOBAL e LOCAL)*; - O não terminal **<table contents source>** gera uma regra de produção com o terminal *(OF)* e três não terminais *<table element list>*, *<user-defined type>* e *<subtable clause>*.

	SQL - 92	SQL - 99	Reflexo
criação de tabela	<table definition> ::= CREATE [{ GLOBAL | LOCAL} TEMPORARY] TABLE <table name> <table element list> [ON COMMIT { DELETE | PRESERVE } ROWS]	<table definition> ::= CREATE [**<table scope>**] TABLE <table name> **<table contents source>** [ON COMMIT **<table commit action>** ROWS]	- *<table element list>* gera uma regra de produção com os três terminais *(left paren, right paren e comma)* e o não terminal *<table element>* (essa regra não teve alteração entre as gramáticas de 92 e 99); - *<table element>* gera uma nova regra com cinco não terminais *<column definition>*, *<table constraint definition>*, e outros três novos (SQL-99): *<like clause>*, *<self-referencing column specification>* e *<column options>*; - *<user-defined type>* gera uma regra com o não terminal *<user-defined type name>*; - *<user-defined type name>* gera uma regra com o não terminal *<schema qualified type name>*; - *<subtable clause>* gera uma regra com o não terminal *(UNDER)* e o não terminal *<supertable clause>*; - *<supertable clause>* gera uma regra com o não terminal *<supertable name>*.

	SQL - 92	SQL - 99	Reflexo
novo tipo de dado	<data type> ::= <character string type> [CHARACTER SET <character set specification>] | <national character string type> | <bit string type> | <numeric type> | <datetime type> | <interval type>	<data type> ::= <predefined type> | <row type> | <user-defined type> | <reference type> | <collection type>	- Organização dos tipos por categoria, com a criação de 5 novas regras de produção. **<predefined type>** ::= <character string type> [CHARACTER SET <character set specification>] | <national character string type> | <binary large object string type> | <bit string type> | <numeric type> | <boolean type> | <datetime type> | <interval type> - O não terminal <*boolean type*> gerou uma regra de produção com o terminal BOOLEAN. - Adição do literal boolean, com os terminais *(TRUE, FALSE e UNKNOWN)*; - Adição da condição de busca, em que gerou uma regra de produção com um não terminal <*boolean value expression*>; - <*row type*> gera uma regra de produção com o terminal *(ROW)* e o não terminal <*row type body*>;

	SQL - 92	SQL - 99	Reflexo
novo tipo de dado	<data type> ::= <character string type> [CHARACTER SET <character set specification>] | <national character string type> | <bit string type> | <numeric type> | <datetime type> | <interval type>	<data type> ::= <predefined type> | <row type> | <user-defined type> | <reference type> | <collection type>	- <*row type body*> gera uma regra de produção com os três terminais *(left paren, right paren e comma)* e o não terminal <*field definition*>. - <*reference type*> gera uma regra de produção com o terminal *(REF, left paren e right paren)* e os dois não terminais <*referenced type*> e <*scope clause*>; - <*referenced type*> gera uma regra de produção com o não terminal <*user-defined type*>; - <*scope clause*> gera uma regra de produção com o terminal *(SCOPE)* e o não terminal <*table name*>; - <*collection type*> gera uma regra de produção com os dois não terminais <*data type*> e <*array specification*>; - <*array specification*> gera uma regra de produção com os terminais *(left bracket, right bracket e trigraph)* e os dois não terminais <*collection type constructor*> e <*unsigned integer*>.

- **Construção do exemplo em SQL 92 com insert**

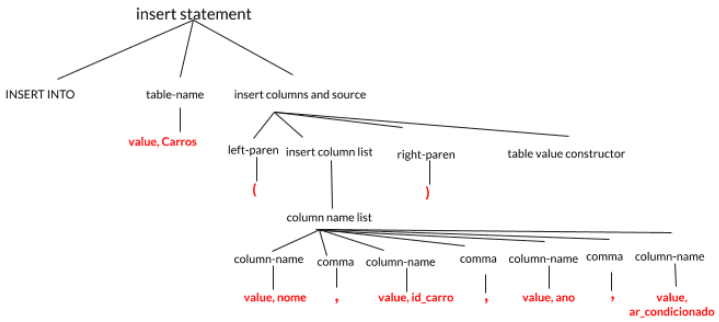
Exemplo 2 na versão SQL 92

SQL 92 - Data Type BIT

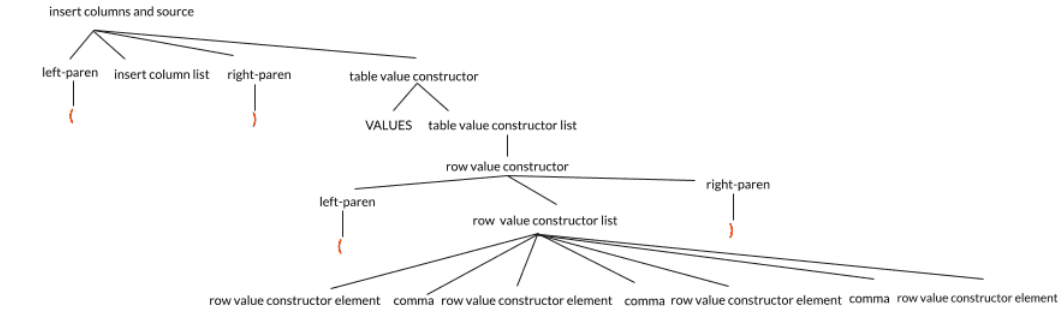
```
INSERT INTO Carros(
        nome,
        id_carro,
        ano,
        ar_condicionado)
VALUES (Fiat Uno,253113,2013,'1')
```

Acesso ao link da gramática de SQL-92 <https://ronsavage.github.io/SQL/sql-92.bnf.html#insert%20statement>.

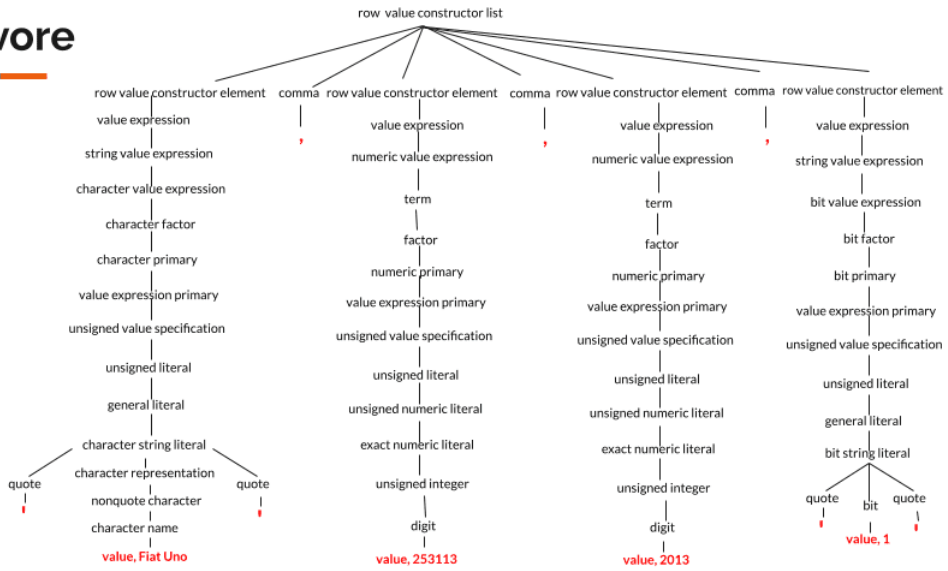
Árvore



Árvore



Árvore



- **Construção do exemplo em SQL 99 com insert**

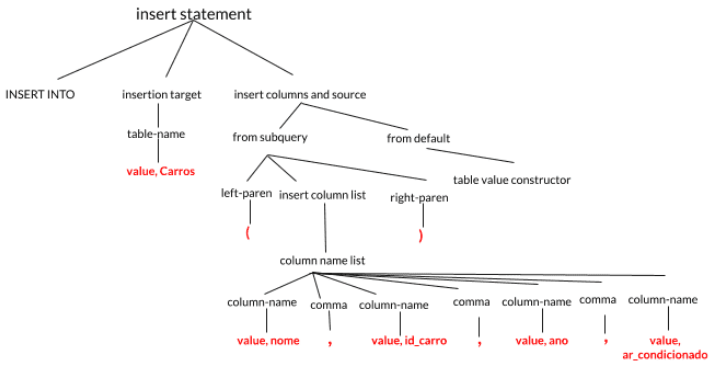
Exemplo 2 na versão SQL 99

SQL 99 - Data Type BOOLEAN

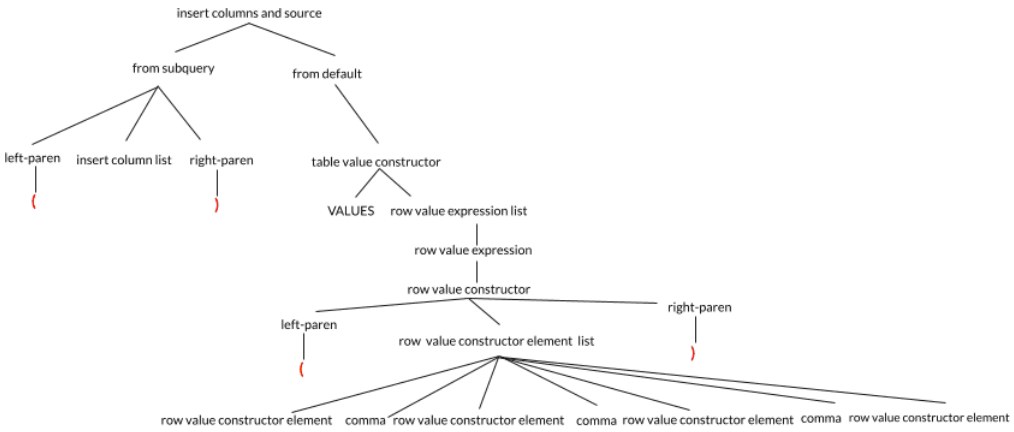
```
INSERT INTO Carros(
        nome,
        id_carro,
        ano,
        ar_condicionado)
VALUES (Fiat Uno,253113,2013,TRUE)
```

Acesso ao link da gramática de SQL-99 <https://ronsavage.github.io/SQL/sql-99.bnf.html#insert%20statement>.

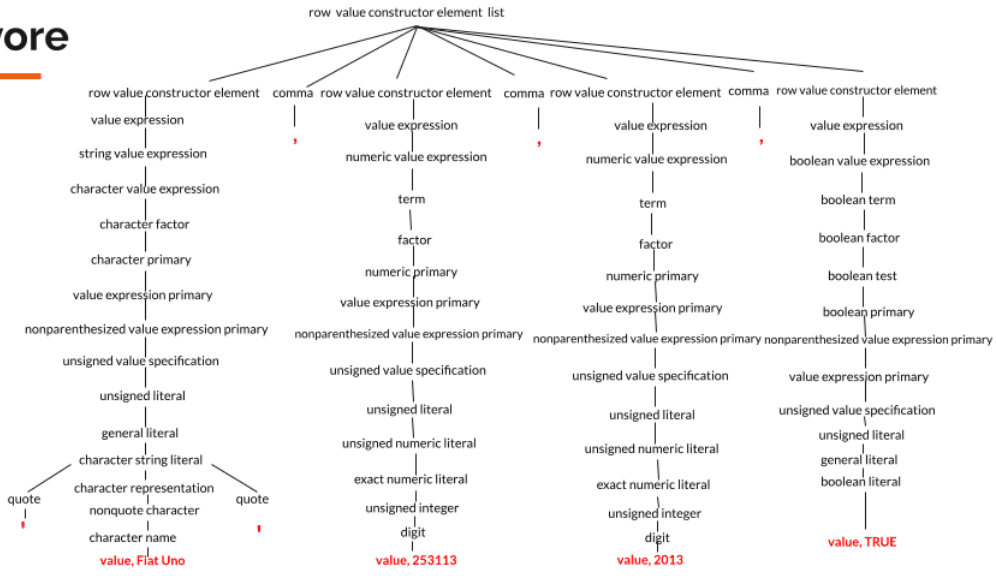
Árvore



Árvore



Árvore



- **Análise das mudanças entre as gramáticas de 92 e 99**

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<insert statement> ::= INSERT INTO <table name> <insert columns and source>	<insert statement> ::= INSERT INTO **<insertion target>** <insert columns and source>	- Adição de 1 nova regra de produção **<insertion target>** ::= <table name> - O não terminal **<table name>** gera uma nova regra de produção contendo o não terminal *<local or schema qualified name>;* - *<local or schema qualified name>* gera uma nova regra de produção com os dois não terminais *<local or schema qualifier>* e *<qualified identifier>* e o terminal *(period)*; - *<local or schema qualifier>* gera uma regra de produção com o não terminal *<schema name>* e terminal *(MODULE)*; - *<qualified identifier>* gera uma regra de produção com o não terminal *<identifier>;*

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<insert statement> ::= INSERT INTO <table name> <insert columns and source>	<insert statement> ::= INSERT INTO **<insertion target>** <insert columns and source>	- Enquanto na gramática de 92, o não terminal <table name> gera a regra de produção contendo os dois não terminais *<qualified name>* e *<qualified local table name>*; - *<qualified name>* gera uma regra de produção com os dois não terminais *<schema name>* e *<qualified identifier>* e o terminal *(period)*; - *<qualified local table name>* gera uma regra com os dois terminais *(MODULE e period)* e o não terminal *<local table name>*.

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<insert columns and source> ::= [<left paren> <insert column list> <right paren>] <query expression> | DEFAULT VALUES	<insert columns and source> ::= **<from subquery> | <from constructor> | <from default>**	- Adição de 3 novas regras de produção **<from subquery>** ::= [<left paren> <insert column list> <right paren>] [<override clause>] <query expression> **<from constructor>** ::= [<left paren> <insert column list> <right paren>] [<override clause>] <contextually typed table value constructor> **<from default>** ::= DEFAULT VALUES - O não terminal **<from subquery>** gera uma nova regra de produção contendo os não terminais *<insert column list>*, *<override clause>* e *<query expression>* e os terminais *(left paren e right paren)*; - *<insert column list>* gera uma regra de produção com um não terminal *<column name list>*;

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<insert columns and source> ::= [<left paren> <insert column list> <right paren>] <query expression> | DEFAULT VALUES	<insert columns and source> ::= **<from subquery> | <from constructor> | <from default>**	- *<override clause>* gera uma nova regra de produção com os terminais *(OVERRIDING USER VALUE e OVERRIDING SYSTEM VALUE)*; - *<query expression>* gera uma nova regra como os não terminais *<with clause>* e *<query expression body>*; - *<with clause>* gera uma nova regra com os terminais *(WITH e RECURSIVE)* e o não terminal *<with list>*; - *<query expression body>* gera uma regra com os não terminais *<non-join query expression>* e *<joined table>*; - Vale salientar que a regra *<query expression body>*, equivale a regra de *<query expression>* na gramática de 92; - Ocorrem distinções das regras *<non-join query expression>* e *<joined table>* entre as gramáticas de 92 e 99, são elas: · adição do terminal *(DISTINCT)* na regra *<non-join query expression>*;

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<insert columns and source> ::= [<left paren> <insert column list> <right paren>] <query expression> | DEFAULT VALUES	<insert columns and source> ::= **<from subquery> | <from constructor> | <from default>**	· diferenças na regra de *<joined table>*, tendo em vista que ocorreu de na gramática de 92, ficar assim: <joined table> ::= <cross join> | <qualified join> | <natural join> | <union join> · enquanto na de 99: <joined table> ::= <cross join> | <qualified join> | <left paren> <joined table> <right paren> - O não terminal **<from constructor>** gera uma nova regra de produção contendo os não terminais *<insert column list>*, *<override clause>* e *<contextually typed table value constructor>* e os terminais *(left paren e right paren)*; - *<contextually typed table value constructor>* gera uma nova regra com o não terminal *<contextually typed row value expression list>* e o terminal *(VALUES)*; - O não terminal **<from default>** gera uma regra de produção contendo os terminais *(DEFAULT e VALUES)*;

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<table value constructor> ::= VALUES <table value constructor list>	<table value constructor> ::= VALUES **<row value expression list>**	- Adição de 1 nova regra de produção: **<row value expression list>** ::= <row value expression> [{ <comma> <row value expression> }...] - O não terminal **<row value expression>** gera uma nova regra de produção contendo os não terminais *<row value special case>* e *<row value constructor>* - *<row value constructor>* gera uma nova regra de produção com os três não terminais *<row value constructor element>*, *<row value constructor element list>* e *<row subquery>* e os terminais *(ROW, left paren e right paren)*; - Na gramática de 92, não apresenta o não terminal **<row value expression>**, sendo essa regra apenas em 99, como pode ser visto a seguir: · <table value constructor> ::= VALUES **<table value constructor list>, vemos o nome da regra diferente da gramática 99.**

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<table value constructor> ::= VALUES <table value constructor list>	<table value constructor> ::= VALUES **<row value expression list>**	· *<table value constructor list>* gera a regra com o não terminal *<row value constructor>* e o terminal *(comma)*. Veja aqui a diferença de 92, onde não existiu o passo anterior com o **<row value expression>**, porém se atentar que foi adicionado em 99 um novo terminal nessa regra o *<row value special case>*. · *<row value constructor>* na gramática de 92 gera uma regra de produção semelhante a de 99, onde possui os três não terminais *<row value constructor element>*, *<row value constructor element list>* e *<row subquery>* com a diferença apenas que tem os terminais *(left paren e right paren)*;

	SQL - 92	SQL - 99	Reflexo
inserir dados na tabela	<value expression primary> ::= <unsigned value specification> | <column reference> |<set function specification> | <scalar subquery> | <case expression> | <left paren> <value expression> <right paren> | <cast specification>	<value expression primary> ::= **<parenthesized value expression>|<nonparenthesized value expression primary>**	- O não terminal **<parenthesized value expression>** gera uma regra com o não terminal *<value expression>* e os terminais *(left paren e right paren)*; - O não terminal **<nonparenthesized value expression primary>** gera uma regra com os não terminais *<unsigned value specification>*, *<column reference>*, *<set function specification>*, *<scalar subquery>*, *<case expression>*, *<cast specification>*, *<subtype treatment>*, *<attribute or method reference>*, *<reference resolution>*, *<collection value constructor>*, *<routine invocation>*, *<field reference>*, *<element reference>*, *<method invocation>*, *<static method invocation>* e *<new specification>*.

Apêndice B

SQL *Reserved words*

	SQL - 92	SQL - 99	Considerações sobre as mudanças
1	ABSOLUTE	ABSOLUTE	
2	ACTION	ACTION	
3	ADD	ADD	
4	--	AFTER	Função de gatilho executada após um término de instrução de disparo (uma inserção, atualização ou exclusão) que é aplicada para uma tabela
5	ALL	ALL	
6	ALLOCATE	ALLOCATE	
7	ALTER	ALTER	
8	AND	AND	
9	ANY	ANY	
10	ARE	ARE	
11	--	ARRAY	Novo tipo de dado

12	AS	AS	
13	ASC	ASC	
14	ASSERTION	ASSERTION	
15	AT	AT	
16	AUTHORIZATION	AUTHORIZATION	
17	AVG	--	Passou a ser non-reserved keyword
18	--	BEFORE	Função de gatilho executada antes um término de instrução de disparo (uma inserção, atualização ou exclusão) que é aplicada para uma tabela
19	BEGIN	BEGIN	
20	BETWEEN	BETWEEN	
21	--	BINARY	Novo tipo de dado
22	BIT	BIT	
23	BIT_LENGTH	--	Passou a ser non-reserved keyword
24	--	BLOB	Novo tipo de dado
25	--	BOOLEAN	Novo tipo de dado
26	BOTH	BOTH	
27	--	BREADTH	Cláusula rápida de percorrer uma árvore ou gráfico com o mínimo de código.
28	BY	BY	
29	--	CALL	Função para executar uma série de instruções em SQL
30	CASCADE	CASCADE	
31	CASCADED	CASCADED	
32	CASE	CASE	
33	CAST	CAST	

34	CATALOG	CATALOG	
35	CHAR	CHAR	
36	CHARACTER	CHARACTER	
37	CHARACTER_LENGTH	--	Passou a ser non-reserved keyword
38	CHAR_LENGTH	--	Passou a ser non-reserved keyword
39	CHECK	CHECK	
40	--	CLOB	Novo tipo de dado
41	CLOSE	CLOSE	
42	COALESCE	--	Passou a ser non-reserved keyword
43	COLLATE	COLLATE	
44	COLLATION	COLLATION	
45	COLUMN	COLUMN	
46	COMMIT	COMMIT	
47	--	CONDITION	Função representar uma condição em SQL
48	CONNECT	CONNECT	
49	CONNECTION	CONNECTION	
50	CONSTRAINT	CONSTRAINT	
51	CONSTRAINTS	CONSTRAINTS	
52	--	CONSTRUCTOR	
53	CONTINUE	CONTINUE	
54	CONVERT	--	Passou a ser non-reserved keyword
55	CORRESPONDING	CORRESPONDING	
56	CREATE	CREATE	
57	CROSS	CROSS	
58	--	CUBE	Função que é uma extensão

			da cláusula GROUP BY. Permite gerar subtotais para todas combinações de colunas de agrupamento especificadas na GROUP BY cláusula
59	CURRENT	CURRENT	
60	CURRENT_DATE	CURRENT_DATE	
61	--	CURRENT_DEFAULT_TRANSFORM_GROUP	Registro especial que especifica um valor que identifica o nome do grupo de transformação usado por instruções SQL dinâmicas para trocar valores fornecidos pelo usuário com programas host
62	--	CURRENT_TRANSFORM_GROUP_FOR_TYPE	
63	--	CURRENT_PATH	Registro especial que identifica o caminho SQL usado ao resolver nomes de funções não qualificados, nomes de procedimentos, nomes de tipos de dados, nomes de variáveis globais e nomes de objetos de módulo em instruções SQL preparadas dinamicamente
64	--	CURRENT_ROLE	
65	CURRENT_TIME	CURRENT_TIME	
66	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	
67	CURRENT_USER	CURRENT_USER	
68	CURSOR	CURSOR	
69	--	CYCLE	
70	--	DATA	Adicionada como reserved word em 99
71	DATE	DATE	
72	DAY	DAY	
73	DEALLOCATE	DEALLOCATE	

74	DEC	DEC	
75	DECIMAL	DECIMAL	
76	DECLARE	DECLARE	
77	DEFAULT	DEFAULT	
78	DEFERRABLE	DEFERRABLE	
79	DEFERRED	DEFERRED	
80	DELETE	DELETE	
81	--	DEPTH	
82	--	DEREF	
83	DESC	DESC	
84	DESCRIBE	DESCRIBE	
85	DESCRIPTOR	DESCRIPTOR	
86	--	DETERMINISTIC	
87	DIAGNOSTICS	DIAGNOSTICS	
88	DISCONNECT	DISCONNECT	
89	DISTINCT	DISTINCT	
90	--	DO	
91	DOMAIN	DOMAIN	
92	DOUBLE	DOUBLE	
93	DROP	DROP	
94	--	DYNAMIC	
95	--	EACH	
96	ELSE	ELSE	
97	--	ELSEIF	
98	END	END	
99	END-EXEC	END-EXEC	
100	--	EQUALS	

101	ESCAPE	ESCAPE	
102	EXCEPT	EXCEPT	
103	EXCEPTION	EXCEPTION	
104	EXEC	EXEC	
105	EXECUTE	EXECUTE	
106	EXISTS	EXISTS	
107	--	EXIT	
108	EXTERNAL	EXTERNAL	
109	EXTRACT	--	Passou a ser non-reserved keyword
110	FALSE	FALSE	
111	FETCH	FETCH	
112	FIRST	FIRST	
113	FLOAT	FLOAT	
114	FOR	FOR	
115	FOREIGN	FOREIGN	
116	FOUND	FOUND	
117	FROM	FROM	
118	--	FREE	
119	FULL	FULL	
120	--	FUNCTION	
121	--	GENERAL	
122	GET	GET	
123	GLOBAL	GLOBAL	
124	GO	GO	
125	GOTO	GOTO	
126	GRANT	GRANT	

127	GROUP	GROUP	
128	--	GROUPING	Usada em conjunto com conjuntos de agrupamento e supergrupos, a função GROUPING retorna um valor que indica se uma linha retornada em um conjunto de respostas GROUP BY é uma linha gerada por um conjunto de agrupamento que exclui a coluna representada por expressão
129	--	HANDLE	
130	HAVING	HAVING	
131	--	HOLD	
132	HOUR	HOUR	
133	IDENTITY	IDENTITY	
134	--	IF	A instrução IF seleciona um caminho de execução com base na avaliação de uma condição
135	IMMEDIATE	IMMEDIATE	
136	IN	IN	
137	INDICATOR	INDICATOR	
138	INITIALLY	INITIALLY	
139	INNER	INNER	
140	--	INOUT	
141	INPUT	INPUT	
142	INSENSITIVE	--	Passou a ser non-reserved keyword
143	INSERT	INSERT	
144	INT	INT	
145	INTEGER	INTEGER	
146	INTERSECT	INTERSECT	

147	INTERVAL	INTERVAL	
148	INTO	INTO	
149	IS	IS	
150	ISOLATION	ISOLATION	
151	JOIN	JOIN	
152	KEY	KEY	
153	LANGUAGE	LANGUAGE	
154	--	LARGE	
155	LAST	LAST	
156	--	LATERAL	
157	LEADING	LEADING	
158	--	LEAVE	
159	LEFT	LEFT	
160	LEVEL	LEVEL	
161	LIKE	LIKE	
162	LOCAL	LOCAL	
163	--	LOCALTIME	
164	--	LOCALTIMESTAMP	
165	--	LOCATOR	
166	--	LOOP	
167	LOWER	--	Passou a ser non-reserved keyword
168	--	MAP	
169	MATCH	MATCH	
170	MAX	--	Passou a ser non-reserved keyword
171	MIN	--	Passou a ser non-reserved keyword
172	--	METHOD	
173	MINUTE	MINUTE	
174	--	MODIFIES	

175	MODULE	MODULE	
176	MONTH	MONTH	
177	NAMES	NAMES	
178	NATIONAL	NATIONAL	
179	NATURAL	NATURAL	
180	NCHAR	NCHAR	
181	--	NCLOB	Novo tipo de dado
182	--	NESTING	
183	--	NEW	
184	NEXT	NEXT	
185	NO	NO	
186	--	NONE	
187	NOT	NOT	
188	NULL	NULL	
189	NULLIF	--	Passou a ser non-reserved keyword
190	NUMERIC	NUMERIC	
191	--	OBJECT	
192	OCTET_LENGTH	--	Passou a ser non-reserved keyword
193	OF	OF	
194	--	OLD	
195	ON	ON	
196	ONLY	ONLY	
197	OPEN	OPEN	
198	OPTION	OPTION	
199	OR	OR	
200	ORDER	ORDER	

201	--	ORDINALITY	
202	--	OUT	
203	OUTER	OUTER	
204	OUTPUT	OUTPUT	
205	OVERLAPS	OVERLAPS	
206	PAD	PAD	
207	--	PARAMETER	
208	PARTIAL	PARTIAL	
209	--	PATH	
210	POSITION	--	Passou a ser non-reserved keyword
211	PRECISION	PRECISION	
212	PREPARE	PREPARE	
213	PRESERVE	PRESERVE	
214	PRIMARY	PRIMARY	
215	PRIOR	PRIOR	
216	PRIVILEGES	PRIVILEGES	
217	PROCEDURE	PROCEDURE	
218	PUBLIC	PUBLIC	
219	READ	READ	
220	--	READS	
221	REAL	REAL	
222	--	RECURSIVE	
223	--	REDO	
224	--	REF	
225	REFERENCES	REFERENCES	
226	--	REFERENCING	
227	RELATIVE	RELATIVE	
228	--	RELEASE	
229	--	REPEAT	

230	--	RESIGNAL	
231	RESTRICT	RESTRICT	
232	--	RESULT	
233	--	RETURN	
234	--	RETURNS	
235	REVOKE	REVOKE	
236	RIGHT	RIGHT	
237	--	ROLE	
238	ROLLBACK	ROLLBACK	
239	--	ROLLUP	
240	--	ROUTINE	
241	--	ROW	
242	ROWS	ROWS	
243	--	SAVEPOINT	Use a instrução SAVEPOINT para definir um ponto de salvamento dentro de uma transação
244	SCHEMA	SCHEMA	
245	SCROLL	SCROLL	
246	--	SEARCH	
247	SECOND	SECOND	
248	SECTION	SECTION	
249	SELECT	SELECT	
250	SESSION	SESSION	
251	SESSION_USER	SESSION_USER	
252	SET	SET	
253	--	SETS	
254	--	SIGNAL	A instrução SIGNAL é usada para sinalizar um erro ou condição de aviso. Isso faz com que um erro ou aviso seja retornado com o SQLSTATE especificado, junto com o texto da

			mensagem opcional
255	--	SIMILAR	
256	SIZE	SIZE	
257	SMALLINT	SMALLINT	
258	SOME	SOME	
259	SPACE	SPACE	
260	--	SPECIFIC	
261	--	SPECIFICTYPE	
262	SQL	SQL	
263	SQLCODE	--	Excluída
264	SQLERROR	--	Excluída
265	--	SQLEXCEPTION	
266	SQLSTATE	SQLSTATE	
267	--	SQLWARNING	
268	--	START	
269	--	STATE	
270	--	STATIC	
271	SUBSTRING	--	Passou a ser non-reserved keyword
272	SUM	--	Passou a ser non-reserved keyword
273	SYSTEM_USER	SYSTEM_USER	
274	TABLE	TABLE	
275	TEMPORARY	TEMPORARY	
276	THEN	THEN	
277	TIME	TIME	
278	TIMESTAMP	TIMESTAMP	
279	TIMEZONE_HOUR	TIMEZONE_HOUR	
280	TIMEZONE_MINUTE	TIMEZONE_MINUTE	
281	TO	TO	

282	TRAILING	TRAILING	
283	TRANSACTION	TRANSACTION	
284	TRANSLATE	--	Passou a ser non-reserved keyword
285	TRANSLATION	TRANSLATION	
286	--	TREAT	
287	--	TRIGGER	Função que cria um gatilho. Geralmente ações que acionam os triggers são alterações nas tabelas por meio de operações de inserção, exclusão e atualização de dados (insert, delete e update).
288	TRIM	--	Passou a ser non-reserved keyword
289	TRUE	TRUE	
290	--	UNDER	
291	--	UNDO	
292	UNION	UNION	
293	UNIQUE	UNIQUE	
294	UNKNOWN	UNKNOWN	
295	--	UNNEST	
296	--	UNTIL	
297	UPDATE	UPDATE	
298	UPPER	--	Passou a ser non-reserved keyword
299	USAGE	USAGE	
300	USER	USER	
301	USING	USING	
302	VALUE	VALUE	
303	VALUES	VALUES	
304	VARCHAR	VARCHAR	
305	VARYING	VARYING	

306	VIEW	VIEW	
307	WHEN	WHEN	
308	WHENEVER	WHENEVER	
309	WHERE	WHERE	
310	--	WHILE	A instrução WHILE repete a execução de uma instrução ou grupo de instruções enquanto uma condição especificada for verdadeira.
311	WITH	WITH	
312	--	WITHOUT	
313	WORK	WORK	
314	WRITE	WRITE	
315	YEAR	YEAR	
316	ZONE	ZONE	

Apêndice C

SQL *Non-reserved words*

	SQL - 92	SQL - 99	Considerações sobre as mudanças
1	--	ABS	
2	ADA	ADA	
3	--	ADMIN	
4	--	ASENSITIVE	
5	--	ASSIGNMENT	
6	--	ASYMMETRIC	
7	--	ATOMIC	
8	--	ATTRIBUTE	
9	--	AVG	Era reserved keyword na gramática de 92
10	--	BIT_LENGTH	Era reserved keyword na gramática de 92
11	C	C	

12	--	CALLED	
13	--	CARDINALITY	
14	CATALOG_NAME	CATALOG_NAME	
15	--	CHAIN	
16	--	CHAR_LENGTH	Era reserved keyword na gramática de 92
17	--	CHARACTERISTICS	
18	--	CHARACTER_LENGTH	Era reserved keyword na gramática de 92
19	CHARACTER_SET_CATALOG	CHARACTER_SET_CATALOG	
20	CHARACTER_SET_NAME	CHARACTER_SET_NAME	
21	CHARACTER_SET_SCHEMA	CHARACTER_SET_SCHEMA	
22	--	CHECKED	
23	CLASS_ORIGIN	CLASS_ORIGIN	
24	--	COALESCE	Era reserved keyword na gramática de 92
25	COBOL	COBOL	
26	COLLATION_CATALOG	COLLATION_CATALOG	
27	COLLATION_NAME	COLLATION_NAME	
28	COLLATION_SCHEMA	COLLATION_SCHEMA	
29	COLUMN_NAME	COLUMN_NAME	
30	COMMAND_FUNCTION	COMMAND_FUNCTION	
31	--	COMMAND_FUNCTION_CODE	
32	COMMITTED	COMMITTED	
33	--	CONDITION_IDENTIFIER	
34	CONDITION_NUMBER	CONDITION_NUMBER	

35	CONNECTION_NAME	CONNECTION_NAME	
36	CONSTRAINT_CATALOG	CONSTRAINT_CATALOG	
37	CONSTRAINT_NAME	CONSTRAINT_NAME	
38	CONSTRAINT_SCHEMA	CONSTRAINT_SCHEMA	
39	--	CONTAINS	
40	--	CONVERT	Era reserved keyword na gramática de 92
41	--	COUNT	
42	CURSOR_NAME	CURSOR_NAME	
43	DATA	--	Adicionada para reserved word em 99
44	DATETIME_INTERVAL_CODE	DATETIME_INTERVAL_CODE	
45	DATETIME_INTERVAL_PRECISION	DATETIME_INTERVAL_PRECISION	
46	DYNAMIC_FUNCTION	--	Excluída
47	--	DEFINED	
48	--	DEFINER	
49	--	DEGREE	
50	--	DERIVED	
51	--	DISPATCH	
52	--	EVERY	
53	--	EXTRACT	Era reserved keyword na gramática de 92
54	--	FINAL	
55	FORTRAN	FORTRAN	
56	--	G	
57	--	GENERATED	
58	--	GRANTED	

59	--	HIERARCHY	
60	--	IMPLEMENTATION	
61	--	INSENSITIVE	Era reserved keyword na gramática de 92
62	--	INSTANCE	
63	--	INSTANTIABLE	
64	--	INVOKER	
65	--	K	
66	--	KEY_MEMBER	
67	--	KEY_TYPE	
68	LENGTH	LENGTH	
69	--	LOWER	Era reserved keyword na gramática de 92
70	--	M	
71	--	MAX	Era reserved keyword na gramática de 92
72	--	MIN	Era reserved keyword na gramática de 92
73	MESSAGE_LENGTH	MESSAGE_LENGTH	
74	MESSAGE_OCTET_LEN GTH	MESSAGE_OCTET_LEN GTH	
75	MESSAGE_TEXT	MESSAGE_TEXT	
76	--	MOD	
77	MORE	MORE	
78	MUMPS	MUMPS	
79	NAME	NAME	
80	NULLABLE	NULLABLE	
81	NUMBER	NUMBER	
82	--	NULLIF	Era reserved keyword na

			gramática de 92
83	--	OCTET_LENGTH	Era reserved keyword na gramática de 92
84	--	ORDERING	
85	--	OPTIONS	
86	--	OVERLAY	
87	--	OVERRIDING	
88	PASCAL	PASCAL	
89	--	PARAMETER_MODE	
90	--	PARAMETER_NAME	
91	--	PARAMETER_ORDINAL_POSITION	
92	--	PARAMETER_SPECIFIC_CATALOG	
93	--	PARAMETER_SPECIFIC_NAME	
94	--	PARAMETER_SPECIFIC_SCHEMA	
95	PLI	PLI	
96	--	POSITION	Era reserved keyword na gramática de 92
97	REPEATABLE	REPEATABLE	
98	--	RETURNED_CARDINALITY	
99	RETURNED_LENGTH	RETURNED_LENGTH	
100	RETURNED_OCTET_LENGTH	RETURNED_OCTET_LENGTH	
101	RETURNED_SQLSTATE	RETURNED_SQLSTATE	
102	--	ROUTINE_CATALOG	
103	--	ROUTINE_NAME	

104	--	ROUTINE_SCHEMA	
105	ROW_COUNT	ROW_COUNT	
106	SCALE	SCALE	
107	SCHEMA_NAME	SCHEMA_NAME	
108	--	SCOPE	
109	--	SECURITY	
110	--	SELF	
111	--	SENSITIVE	
112	SERIALIZABLE	SERIALIZABLE	
113	SERVER_NAME	SERVER_NAME	
114	--	SIMPLE	
115	--	SOURCE	
116	--	SPECIFIC_NAME	
117	--	STATEMENT	
118	--	STRUCTURE	
119	--	STYLE	
120	SUBCLASS_ORIGIN	SUBCLASS_ORIGIN	
121	--	SUBSTRING	Era reserved keyword na gramática de 92
122	--	SUM	Era reserved keyword na gramática de 92
123	--	SYMMETRIC	
124	--	SYSTEM	
125	TABLE_NAME	TABLE_NAME	
126	--	TOP_LEVEL_COUNT	
127	--	TRANSACTIONS_COMMITTED	
128	--	TRANSACTIONS_ROLLE	

		D_BACK	
129	--	TRANSACTION_ACTIVE	
130	--	TRANSFORM	
131	--	TRANSFORMS	
132	--	TRANSLATE	Era reserved keyword na gramática de 92
133	--	TRIGGER_CATALOG	
134	--	TRIGGER_SCHEMA	
135	--	TRIGGER_NAME	
136	--	TRIM	Era reserved keyword na gramática de 92
137	TYPE	TYPE	
138	UNCOMMITTED	UNCOMMITTED	
139	UNNAMED	UNNAMED	
140	--	UPPER	Era reserved keyword na gramática de 92

Apêndice D

SQL *Terminals*

	SQL - 92	SQL - 99	Terminal
1	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	simple Latin upper case letter
2	a b c d e f g h i j k l m n o p q r s t u v w x y z	a b c d e f g h i j k l m n o p q r s t u v w x y z	simple Latin lower case letter
3	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	Digit
4	[		**<SQL embedded language character>** -> left bracket
5	]		**<SQL embedded language character>** -> right bracket

Apêndice E

SQL *Special Characters*

	SQL - 92	SQL - 99	Caractere	Considerações sobre as mudanças
1			Espaço	
2	“	“	Dupla aspas	
3	%	%	Módulo, porcento	
4	&	&	E comercial	
5	'	'	Aspa	
6	(	(	parêntese esquerdo	
7	)	)	parêntese direito	
8	*	*	asterisco, multiplicação	
9	+	+	Adicionar, adição unária, concatenação de string	
10	,	,	Vírgula	
11	-	-	Subtrair, menos unário	

12	.	.	Ponto	
13	/	/	Dividir, barra	
14	:	:	Dois pontos	
15	;	;	Ponto e vírgula	
16	<	<	Operador menor que	
17	=	=	Igual	
18	>	>	Operador maior que	
19	?	?	Interrogação	
20		[	Colchete esquerdo	Definido na gramática de 92 pelo não terminal **<SQL embedded language character>**
21		]	Colchete direito	Definido na gramática de 92 pelo não terminal **<SQL embedded language character>**
22		^	Circunflexo	
23	_	_	Sublinhado	
24	|	|	Barra vertical	
25		{	Chave esquerda	
26		}	Chave direita	