USO DE AUTÔMATOS E LINGUAGENS REGULARES PARA PREVENÇÃO DE ATAQUES EM BANCO DE DADOS SQL E NOSQL

JOÃO MIGUEL DE ALBUQUERQUE FARIA[1]
PEDRO VINÍCIUS MESSIAS DELGADO[2]
MARCELO ARTHUR DOS SANTOS MOURA[3]
LUCAS CASSIANO CÉSAR[4]
BRUNNO WAGNER LEMOS DE SOUZA[5]

RESUMO

O crescimento exponencial do volume de dados intensificou o uso de bancos SQL e NoSQL, que, apesar de eficientes, permanecem vulneráveis a ataques de injeção (Souza et al., 2025). Destacam que soluções de segurança devem ir além da simples detecção de ataques, sendo necessárias abordagens capazes de lidar com padrões complexos e responder com eficiência e escalabilidade diante das novas ameaças. Este trabalho propõe uma abordagem baseada na Teoria da Computação, utilizando autômatos finitos e linguagens regulares para detectar e prevenir injeções maliciosas em bancos de dados. A pesquisa, de caráter exploratório e aplicado, emprega modelagem formal e simulações no software JFLAP para avaliar o desempenho do modelo em cenários controlados. O objetivo é desenvolver um modelo computacional que utilize autômatos e expressões regulares para fortalecer a segurança de bancos SQL e NoSQL, oferecendo uma solução eficiente, verificável e escalável.

Palavras-chave: Segurança em Banco de Dados, Autômatos Finitos, Linguagens Regulares.

[1] Bacharelando em Engenharia de Software. UPE – Garanhuns. joao.miguelfaria@upe.br
[2] Bacharelando em Engenharia de Software. UPE – Garanhuns. pedro.delgado@upe.br
[3] Bacharelando em Engenharia de Software. UPE – Garanhuns. marcelo.asmoura@upe.br
[4] Bacharelando em Engenharia de Software. UPE – Garanhuns. lucas.cassiano@upe.br
[5] Doutor em Ciência da Computação. UPE – Garanhuns. brunno.souza@upe.br

USE OF AUTOMATA AND REGULAR LANGUAGES FOR THE PREVENTION OF ATTACKS ON SQL AND NOSQL DATABASES

ABSTRACT

The exponential growth of data volume has intensified the use of SQL and NoSQL databases, which, despite being efficient, remain vulnerable to injection attacks (Souza et al., 2025) . Emphasize that security solutions must go beyond mere attack detection, requiring approaches capable of handling complex patterns and responding efficiently and scalably to emerging threats. This work proposes an approach based on Computation Theory, using finite automata and regular languages to detect and prevent malicious injections in databases. The exploratory and applied research employs formal modeling and simulations in JFLAP software to evaluate the model's performance in controlled scenarios. The objective is to develop a computational model leveraging automata and regular expressions to enhance the security of SQL and NoSQL databases, providing an efficient, verifiable, and scalable solution.

Keywords: *Database Security, Finite Automata, Regular Languages.*

1. INTRODUÇÃO

O volume de dados gerados diariamente cresce de maneira exponencial, muito devido ao aumento do uso de aparelhos eletrônicos. Como dito no artigo “Estudo comparativo entre os modelos de banco de dados NoSQL e SQL”, devido à expansão acelerada da utilização da rede mundial de computadores e ao uso de aparelhos eletrônicos, têm gerado um volume cada vez maior de dados, complexo e não estruturados (Souza; Boer, 2023). Dessa forma, os bancos de dados estão sendo cada vez mais utilizados como alternativa para manipulação desses dados.

Ao falarmos dos bancos de dados, vale ressaltar, que os bancos de dados relacionais (SQL), por conta da sua longa tradição e maturidade em mecanismos de proteção, mantêm esquemas rígidos que podem dificultar a adaptação e a escalabilidade em cenários dinâmicos. Em contrapartida, os bancos NoSQL oferecem modelos de dados flexíveis e prototipação rápida, mas ampliam as superfícies de ataque por carecerem de validações estruturais nativas (Khan et al., 2023). “O crescimento exponencial dos bancos de dados NoSQL, impulsionado por sua capacidade de lidar com grandes volumes de dados não estruturados e escalabilidade horizontal, contrasta com a robustez e integridade transacional dos bancos de dados relacionais (SQL)". Os mesmos autores enfatizam que "apesar da flexibilidade e da eficiência oferecidas pelos sistemas NoSQL, eles apresentam novas ameaças de segurança, especialmente relacionadas a ataques de injeção, que podem comprometer a integridade e a confidencialidade dos dados" (Awad; Cardoso; Bussador, 2024). Vale ressaltar, que os bancos de dados relacionais e não relacionais enfrentam desafios distintos, porém igualmente significativos, quando se fala de segurança.

Por isso, tal pauta se tornou motivo de preocupação. Os ataques de injeção de código, ou seja, um invasor injeta dados maliciosos externos em um sistema vulnerável, tanto SQL quanto NoSQL, representam ameaças persistentes e devastadoras à integridade, confidencialidade e a disponibilidade dos dados corporativos. Com isso, ao analisarmos a vulnerabilidade da segurança em

bancos de dados NoSQL e SQL, com ênfase nos ataques de injeção de código, conseguimos perceber os desafios enfrentados por sistemas amplamente adotados em aplicações web e na nuvem (Awad; Cardoso; Bussador, 2024).

De acordo com estudos recentes, em 2023, aproximadamente 65% das violações de segurança em aplicativos web foram atribuídas a ataques de injeção SQL, evidenciando a magnitude e a frequência deste problema (Colen, 2024). No Brasil, o cenário é igualmente preocupante: o país registrou cerca de 23 bilhões de incidentes cibernéticos no primeiro semestre de 2023, figurando como o país mais visado na América Latina, enquanto dados apontam que "cerca de 23% das vulnerabilidades críticas encontradas em 2023 estavam relacionadas a injeções", o que inclui tanto SQL quanto NoSQL (InovTI, 2023).

Neste contexto, surge a necessidade de abordagens inovadoras e com fundamentação teórica para a detecção e prevenção de ataques de injeção. A Teoria da Computação, especificamente através dos conceitos de autômatos finitos e linguagens formais, oferece um arcabouço matemático robusto para modelar, simular e prevenir estes ataques (Breve, 2009). Autômatos Finitos Determinísticos surgem como ferramentas poderosas para validação de entradas, permitindo a verificação sistemática de padrões maliciosos antes que possam impactar o banco de dados (Awad; Cardoso; Bussador, 2024). Isso acontece, pois, expressões regulares, intimamente relacionadas aos autômatos finitos, possibilitam a especificação precisa de padrões de ataque e a implementação de mecanismos de filtragem eficientes.

A motivação para empregar autômatos e linguagens formais na prevenção de ataques de injeção fundamenta-se em diversos aspectos. Primeiro, estes modelos matemáticos permitem uma análise formal e verificável dos padrões de entrada, oferecendo garantias teóricas sobre a correção da validação (InovTI, 2023). Segundo a eficiência computacional dos autômatos finitos possibilita a implementação de sistemas de detecção em tempo real, com tempo de resposta compatível com as exigências de aplicações de alta performance. Como observado por Souza et al., "soluções de segurança precisam, para além da eficiência na detecção, satisfazer requisitos de tempo de resposta e

escalabilidade" (Souza et al., 2025). Terceiro, Expressões regulares permitem fazer moldes nas regras de busca em diferentes tipos de dados, de forma clara e com fácil manutenção. Novos padrões de dados sensíveis podem ser adicionados ou ajustados alterando-se apenas a lista de expressões, sem impactar o núcleo do programa (Silva; Santos, 2025).

Com isso, este trabalho propõe responder à seguinte questão: como a Teoria da Computação, especificamente o uso de autômatos finitos e linguagens regulares, podem ser aplicados na prevenção de ataques de injeção em bancos de dados SQL e NoSQL?

Nesse sentido, o objetivo principal deste artigo é apresentar uma abordagem de prevenção fundamentada na Teoria da Computação, utilizando autômatos finitos e linguagens regulares como base formal para a detecção e redução de padrões maliciosos. A partir disso, a integração desses conceitos teóricos com implementações práticas visa oferecer mecanismos de proteção eficientes, verificáveis e adaptáveis às formas de ataque.

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os principais conceitos relacionados aos bancos de dados e às vulnerabilidades associadas a ataques de injeção, fornecendo o embasamento teórico necessário para o desenvolvimento desta pesquisa. Inicialmente, são abordadas as características e estruturas dos bancos de dados SQL e NoSQL. Em seguida, são discutidos os ataques de injeção em ambos os contextos, SQL e NoSQL, evidenciando como falhas na validação de entradas podem comprometer a integridade e a segurança das aplicações. Por fim, são apresentados métodos e ferramentas de prevenção e mitigação dessas vulnerabilidades, contemplando soluções tecnológicas voltadas à proteção de sistemas contra esse tipo de ameaça.

2.1 Banco de dados SQL

Os bancos de dados SQL (*Structured Query Language*), foram propostos por Edgar Codd em 1970. Esse modelo possibilita o relacionamento dos dados por meio de tabelas compostas por colunas e linhas, nas quais cada tabela representa uma entidade, as colunas representam os atributos também chamados de campos da relação e as linhas contém registros individuais. Além disso, o modelo relacional permite descrição natural e estruturada dos dados. Codd mostrou que esse modelo devido às suas características de completude e consistência é considerado uma excelente opção de modelagem de banco de dados (Lages; Pereira, 2021). Até os dias atuais, o modelo relacional é amplamente utilizado devido à sua eficiência no gerenciamento de dados relacionais. Os bancos de dados relacionais utilizam a linguagem declarativa SQL, que permite a manipulação, seleção e descrição dos dados de forma padronizada e eficiente. Além disso, os bancos de dados SQL destacam-se por suas propriedades de integridade de dados, processamento de transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e alta capacidade de consulta eficiente (Jatana et al., 2012).

O SQL é uma linguagem declarativa, pois o usuário especifica o que deseja que o computador execute em vez de como a operação deve ser realizada. Essa linguagem permite criar, alterar, recuperar e gerenciar informações de forma padronizada e eficiente, por meio de comandos claros, como *SELECT, INSERT, UPDATE* e *DELETE*, sendo hoje o principal padrão de comunicação para sistemas que usam o modelo relacional. Entretanto, os bancos de dados relacionais apresentam limitações quando aplicados à manipulação de grandes números de dados não estruturados e à escalabilidade horizontal, o que dificulta a distribuição de carga de dados entre múltiplos servidores, ou seja, esse modelo se torna menos adequado para aplicações modernas que exigem processamento massivo de dados e alta escalabilidade.

2.2 Banco de dados NoSQL

Os Bancos de dados NoSQL (*Not Only SQL*) são sistemas não relacionais e relativamente recentes, eles foram projetados para lidar com grandes volumes de dados não estruturados ou semiestruturados com foco em escalabilidade e alto desempenho garantidos através de uma maior flexibilidade de modelagem. O paradigma dos bancos de dados NoSQL engloba uma família de modelos de dados não relacionais, sendo eles: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos (Frozza; Schreiner; Mello, 2022). Diferentemente dos bancos de dados relacionais tradicionais, os sistemas NoSQL não exigem esquemas fixos e costumam ser distribuídos, o que proporciona eficiência no armazenamento, gerenciamento e indexação de conjuntos de dados de tamanho arbitrariamente grande, ao mesmo tempo em que oferece suporte a um grande número de solicitações de usuários simultâneos. O surgimento de bancos de dados NoSQL representou uma resposta às limitações dos sistemas relacionais, surgindo como uma alternativa eficaz para o armazenamento e processamento de grandes volumes de dados em contextos modernos, como aplicações web em larga escala e sistemas de Big Data (Moniruzzaman; Hossain, 2013).

2.3 Ataques de injeção em Bancos SQL

O ataque de injeção em banco de dados SQL é uma das principais ameaças à segurança de sistemas, principalmente de aplicações web. Esses ataques consistem na inserção maliciosa de comandos SQL em campos de entrada, esse tipo de ataque pode ser executado facilmente em qualquer formulário de login, cadastro, ou campo de busca de uma aplicação, com o objetivo de manipular a consulta ao banco de dados para obter acesso não autorizado, modificar ou apagar informações. Essas vulnerabilidades exploram a falta de validação adequada dos dados fornecidos pelo usuário, permitindo que comandos SQL sejam executados arbitrariamente (Lages; Pereira, 2021). Os impactos de uma injeção SQL bem-sucedida podem variar desde o acesso

indevido a informações sensíveis, alteração ou exclusão de dados, controle total do servidor de banco de dados e, em casos extremos, execução de comandos no sistema operacional. Dentre os tipos mais comuns de ataques de injeção SQL se destacam os ataques de tautologia em que o atacante insere condições sempre verdadeiras, ataques de consultas adicionais (*piggyback queries*) nas quais múltiplos comandos são concatenados em uma mesma entrada para executar instruções indesejadas, a injeção por *UNION* que usa o operador *UNION* para combinar resultados de consultas não autorizadas (Silva; Santos, 2025).

2.4 Ataques de injeção em Bancos NoSQL

Com o crescimento dos bancos de dados NoSQL, muitos dos quais usam formatos diferentes de consultas, geralmente baseados em JSON ou documentos, como o MongoDB que é nomeado como o banco de dados NoSQL mais popular de acordo com o rastreamento do DB-Engines plataforma que fornece uma visão abrangente e atualizada sobre a popularidade e uso de sistemas de gerenciamento de banco de dados. Embora as injeções NoSQL ocorram de forma diferente das tradicionais injeções SQL, elas podem ser igualmente prejudiciais. Esses ataques surgiram como forma de explorar a ausência de filtros e validação em consultas NoSQL. Apesar de que os ataques de injeção NoSQL tendem a ocorrer com uma menor taxa porque as consultas que são processadas na linguagem PHP uma linguagem de script, de código aberto amplamente utilizada para o desenvolvimento web dinâmico, as consultas a partir de strings tradicionais são convertidas em objetos antes de serem executadas. Os ataques mais comuns em bancos NoSQL incluem a injeção de operadores, por exemplo $ne, $gt, $regex e $in que podem modificar a lógica de filtragem para contornar autenticações ou recuperar conjuntos de dados não autorizados, outro ataque bastante comum é a manipulação da sintaxe JSON para provocar erros, respostas inesperadas ou revelação de dados por meio de comportamentos diferenciados da aplicação. Por exemplo, a inserção de um operador como {"senha": {"$ne": null}} numa query de autenticação mal construída

pode fazer com que a condição verifique simplesmente por registros cujo campo senha não seja nulo, potencialmente retornando usuários válidos sem que credenciais corretas tenham sido fornecidas (Moniruzzaman; Hossain, 2013).

2.5 Métodos e ferramentas para prevenção e mitigação de ataques de injeção

Ataques de injeção, tanto em bancos SQL quanto NoSQL, exploram a confiança excessiva em dados de entrada sem validação adequada. Embora os bancos NoSQL tenham estruturas diferentes, as vulnerabilidades fundamentais são similares, em ambos os casos, falhas na filtragem e validação podem permitir que comandos indesejados sejam executados no banco, comprometendo a segurança da aplicação. Portanto, a adoção combinada de práticas de segurança tradicionais como higienização da entrada ou até mesmo de abordagem mais complexas como o uso de autômato finito determinístico (DFA) que é proposto como um mecanismo formal de validação capaz de identificar e bloquear padrões de entrada potencialmente maliciosos em banco de dados NoSQL e SQL (Jatana et al., 2012).

O DFA é um modelo matemático amplamente utilizado na teoria da computação para o reconhecimento de linguagens regulares,que são conjuntos de cadeias de caracteres definidos por regras formais ou expressões regulares. Linguagens regulares permitem descrever padrões previsíveis de entradas válidas, como estruturas de consultas ou sequências permitidas de símbolos. Ao definir um conjunto de estados válidos e transições determinísticas entre eles, o autômato permite identificar sequências de entradas que desviam de comportamento esperado, como por exemplo a inserção de operadores lógicos, símbolos especiais ou estruturas de consulta não permitidas (Frozza; Schreiner; Mello, 2022). Assim, linguagens regulares, juntamente com DFAs, fornecem uma base teórica que pode ser aplicada na análise sintática de consultas SQL e NoSQL auxiliando na prevenção de ataques de injeção em sistemas de banco de dados.

3. METODOLOGIA

A metodologia adotada no presente trabalho caracteriza-se como uma pesquisa de natureza exploratória com finalidades aplicacionais, pois busca investigar, modelar e aplicar conceitos de teoria da computação, autômatos finitos e linguagem formal, em um contexto prático de segurança da informação, atrelado à prevenção de ataques de injeção em bancos de dados SQL e NoSQL (Awad; Cardoso; Bussador, 2024). Refere-se também a uma pesquisa com características experimentais, devido a construção e execução de experimentos controlados em ambientes de teste, utilizando ferramentas de simulação como o JFLAP (Java Formal Languages and Automata Package) para a criação e verificação de autômatos (Adithi et al., 2015).

A abordagem metodológica proposta combina técnicas qualitativas e quantitativas. No aspecto qualitativo, realiza-se uma análise dos comportamentos de ataque e defesa, considerando os diversos tipos de injeção nos diferentes bancos de dados (SQL e NoSQL), bem como a modelagem teórica dos padrões semânticos e sintáticos dessas inseguranças (Detecting and defeating SQL injection attacks, 2011). Enquanto no quantitativo são mensurados o desempenho e a eficiência do modelo proposto com base em métricas objetivas como, taxa de detecção, tempo de resposta, precisão e ocorrência de falsos positivos e negativos. Para revisão e aprofundamento dos conteúdos abordados, como também para garantir a relevância e coesão desses assuntos, a pesquisa bibliográfica para esse trabalho foi guiada por meio de buscas estruturadas em bases de dados científicas, como o Google Scholar e o IEEE Xplore. Nessas bases foram utilizadas strings de busca amplas que juntam termos relacionados à teoria de autômatos e segurança da informação. As principais strings de busca utilizadas foram:

- "autômato" OR "autômatos finitos" OR "máquina de estados finitos" OR "autômato finito" OR "linguagens regulares" OR "expressões regulares" OR regex) AND ("segurança digital" OR "cibersegurança" OR "segurança da

informação" OR "segurança de redes" OR "detecção de intrusão" OR "simulação de ataque" OR "simulação de ameaças" OR "defesa"

- "automata" OR "finite automata" OR "finite-state machine" OR "FSM" OR "regular languages" OR "regular expressions" OR "regex") AND ("cybersecurity" OR "cyber security" OR "information security" OR "network security" OR "intrusion detection" OR "attack simulation" OR "threat simulation" OR "penetration testing" OR "defense"

Tabela 1 - Definição da classificação da pesquisa

Classificação da Pesquisa	Tipo de Pesquisa
Quanto à finalidade	Pesquisa aplicada
Quanto à natureza	Pesquisa experimental
Quanto à forma de abordagem	Pesquisa qualitativa e quantitativa
Quanto aos objetivos	Pesquisa exploratória
Quanto aos procedimentos técnicos	Pesquisa bibliográfica

Fonte: Os autores (2025).

3.1 Modelagem e Implementação dos Autômatos

Essa etapa compreende a modelagem formal e implementação prática dos autômatos e expressões regulares. No processo de modelagem, será dada prioridade à utilização de autômatos determinísticos (DFA), devido à sua eficiência computacional e facilidade de implementação, especialmente em cenários que exigem verificação em tempo real de entradas de usuários (Sun; Valgenti; Kim, 2011). Serão projetados autômatos, capazes de reconhecer

padrões válidos (consultas seguras) e rejeitar cadeias que apresentem características de ataques de injeção, como tautologias, operadores indevidos, concatenação de comandos e operadores específicos do ambiente NoSQL. Toda a modelagem será realizada no JFLAP. Neste ambiente de modelagem e testes, as entradas simuladas de usuários serão recebidas e processadas, cada consulta passará, pelo filtro do autômato e pelas validações definidas pelas linguagens dos modelos, permitindo assim a detecção de padrões suspeitos. Esse procedimento possibilita medir a capacidade do modelo de identificar entradas maliciosas.

3.2 Execução dos Experimentos e Análise dos Resultados

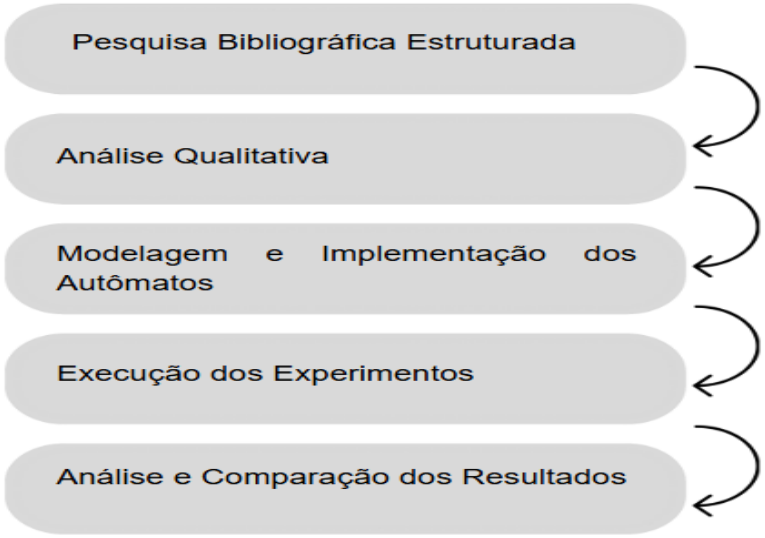
Essa fase consiste na realização dos experimentos e análise dos resultados. Para isso, serão testadas consultas tanto legítimas quanto maliciosas em simulações no JFLAP. Durante os testes serão inseridas consultas legítimas e maliciosas, cada entrada será avaliada pelo sistema de detecção, registrando se a entrada é maliciosa ou não. Cada simulação permitirá analisar como os autômatos reagem a diferentes entradas. Com base nessas observações, será possível discutir o desempenho dos modelos em termos de eficiência na detecção de ataques em banco de dados SQL e NoSQL.

3.3 Aprimoramento, Validação e Limitações

A etapa final engloba o aprimoramento e a validação do modelo proposto, a partir dos resultados experimentais obtidos. Com base nas métricas e nos casos de erro observados, serão realizados ajustes nos autômatos e nas expressões regulares, com o objetivo de reduzir falsos positivos e negativos. A validação cruzada será conduzida utilizando um conjunto de sequências de entradas maliciosas simuladas para testar a eficácia do modelo, e consultas legítimas, de modo a garantir a robustez do modelo em diferentes cenários. Contudo, reconhecem-se algumas limitações em relação ao estudo. O uso do JFLAP, embora adequado para simulações acadêmicas, apresenta restrições de

desempenho e escalabilidade quando aplicado a sistemas reais. Ainda assim, o trabalho busca oferecer uma contribuição relevante e reprodutível ao propor a aplicação de fundamentos da Teoria da Computação na mitigação de vulnerabilidades em bancos de dados SQL e NoSQL.

Figura 1 - Fluxograma Metodológico



Fonte: Os autores (2025).

4. PROPOSTA

A presente pesquisa tem como objetivo o desenvolvimento de modelos de Autômatos Finitos Determinísticos para detecção e prevenção de ataques de injeção em banco de dados SQL e NoSQL, para isso foram escolhidos os ataques mais comuns que ocorrem contra banco de dados, ataques union em banco de dados SQL e tautologias em banco de dados NoSQL. A proposta visa definir a capacidade desses autômatos reconhecerem e validarem padrões em cadeias de caracteres, que, no contexto de segurança dos bancos de dados representam as *queries* de entrada.

4.1 Modelagem do autômato para identificação da palavra UNION como subpalavra em strings

Para detectar a ocorrência da palavra UNION em qualquer posição dentro de uma string de entrada, desenvolveu-se um AFD capaz de reconhecer essa sequência específica como subpalavra. O autômato foi modelado de forma que, a cada caractere lido, ele avance progressivamente pelos estados que representam o prefixo já identificado, retornando para estados anteriores sempre que a sequência esperada for interrompida. Dessa forma, o AFD consegue percorrer toda a entrada sem a necessidade de reiniciar a leitura, garantindo eficiência na verificação. O autômato criado para esta identificação é chamado de M, ele é definido formalmente pela quíntupla M = (Q, Σ, δ, q0, F) onde:

- Q - É um conjunto finito de estados possíveis não vazios do autômato;
- Σ - É um Alfabeto de símbolos de entrada;
- δ - Função de Transição
- qo - É o Estado inicial, tal que qo ∈ Q.
- F - É o conjunto de Estados Finais ou Estados de Aceitação, tal que F ⊆ Q.

O principal objetivo do autômato M é garantir que a sequência de entrada não contenha a palavra “union” reconhecendo assim um ataque potencial. Para a simulação será considerado que as *strings* que o autômato terá acesso serão tratadas e virão em minúsculo como ocorre em sistemas reais, para isso M será definido como:

- M = ({q0, q1, q2, q3, q4, q5}, {a, b, c,..., z}, δ, q0, {q5})
- Q = {q0, q1, q2, q3, q4, q5}
- Σ = {a, b, c, ..., z}
- δ = (estado atual, entrada) = próximo estado
- q0 = estado inicial
- F = {q5}

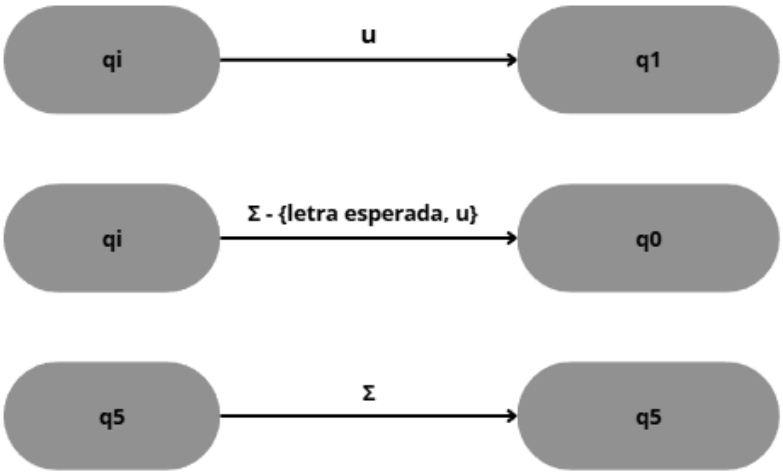
O Autômato M, incorpora regras de transição que garantem o reconhecimento preciso da subpalavra “union” em qualquer parte da *query* de entrada. Portanto as transições especiais do modelo M são definidas da seguinte forma:

- Ação da letra “u”: Para todos os estados qi, onde i < 5, ou seja, de q0 a q4, a leitura da letra “u” força o autômato a recomeçar o reconhecimento da sequência “nion” a partir do segundo caractere, levando-o diretamente ao estado q1. Se por exemplo o autômato está no estado q3 e a próxima entrada for “u” o autômato retorna para o estado q1.
- Ação de retorno ao estado inicial: Qualquer letra do alfabeto que não seja o esperado para dar continuidade a sequência “union” e que não seja a letra “u” leva o autômato de volta ao estado q0 (estado inicial), garantindo que o modelo fique no estado q0 até encontrar o primeiro caractere da palavra “union”.
- Regra do estado final: Uma vez que o autômato esteja no estado q5 (estado final), onde a palavra “union” foi reconhecida, qualquer letra lida em seguida mantém o modelo no estado final q5. Fazendo com que, a palavra “union” seja reconhecida independentemente dos caracteres que a precedem ou a sucedem.

Dessa forma, a linguagem reconhecida, chamada de L, pelo autômato M é o conjunto de todas as palavras que possuem a subpalavra “union” em qualquer posição

- $L = \{x \in \Sigma \mid x \text{ contém a subcadeia “union”}\}$

Figura 2 - Resumo das Regras de Funcionamento do Autômato M



Fonte: Os autores (2025).

Para um melhor entendimento do funcionamento do autômato M, agora será apresentada a tabela de estados que define as transições como δ = (estado atual, entrada) = próximo estado.

Tabela 2 - Tabela de Transição do Autômato M

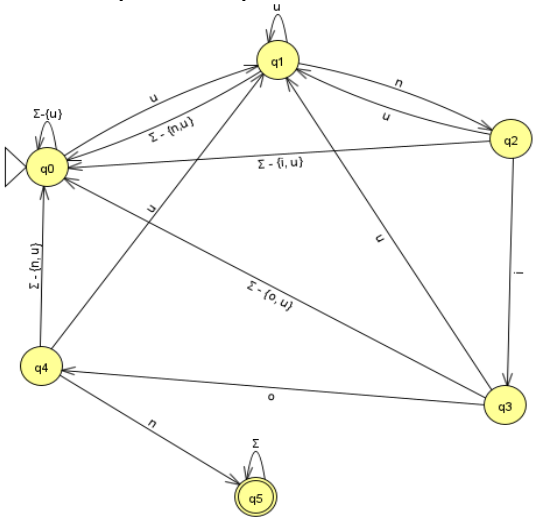
Estado Atual	Entrada	Próximo Estado
q0	u	q1
q0	Σ - {u}	q0
q1	n	q2
q1	u	q1
q1	Σ - {n, u}	q0
q2	i	q3
q2	u	q1

q2	Σ - {i, u}	q0
q3	o	q4
q3	u	q1
q3	Σ - {o, u}	q0
q4	n	q5
q4	u	q1
q4	Σ - {n, u}	q0
q5	Σ	q5

Fonte: Os autores (2025).

Onde o símbolo Σ representa o alfabeto usado pelo modelo e a notação Σ - {letra} representa o alfabeto usado menos o caractere ou o conjunto de caracteres presentes na chave. Como resultado a seguir, é apresentada a imagem da modelagem compactada do autômato M. Esta versão prioriza a legibilidade e a eficiência visual, usando símbolos para representar a volta ao estado inicial e o loop no estado final.

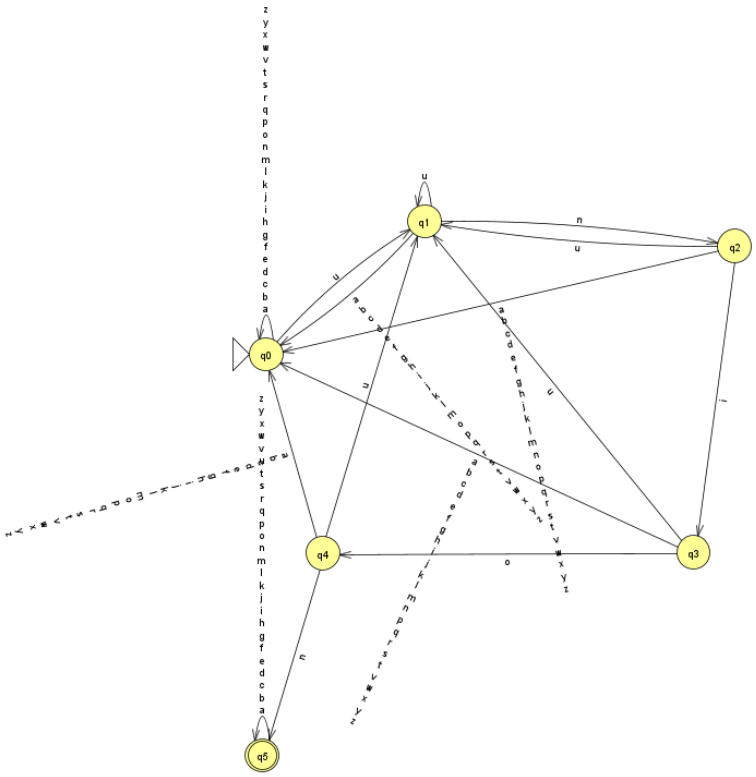
Figura 3 - Modelagem Compactada para Melhor Visualização do Autômato M



Fonte: Os autores (2025).

A seguir é mostrada a imagem da modelagem completa e original do autômato M. Mostrando cada transição para cada símbolo do alfabeto e regras de transições adotadas.

Figura 4 - Modelagem Original do Autômato M



Fonte: Os autores (2025).

A validação do autômato M foi realizada através de testes de reconhecimento em um conjunto de seis strings de entrada, com 3 delas contendo a palavra “union” como subpalavra e 3 delas não. O objetivo foi confirmar se as entradas com a subpalavra “union” eram tratadas como maliciosas e as que não tivessem, não fossem tratadas como maliciosas.

Tabela 3 - Tabela de Resultados do Autômato M

String	Tipo	Identificada como Maliciosa
selectnomeunionselectsenha	Maliciosa	Sim
nomedoprojeto	Não Maliciosa	Não
usuariooruniuunionselectsenhadatabase	Maliciosa	Sim
nomedousuario	Não Maliciosa	Não
usernameunionselectusernamesenhadatabase	Maliciosa	Sim
nomedaconta	Não Maliciosa	Não

Fonte: Os autores (2025).

Como conclusão dos resultados tem-se que eles confirmaram o comportamento esperado do modelo M, fazendo com que ele identificasse as strings que contenham a subpalavra “union”. Confirmando assim que o autômato consegue identificar a palavra union em uma string contínua.

4.2 Modelagem do autômato para identificação de ataques UNION

A detecção de ataques de injeção SQL, principalmente aqueles que exploram o uso de UNION para roubar dados, requer um mecanismo de validação sintática robusto e determinístico. O AFD neste contexto se estabelece como o modelo ideal para esta finalidade, por conta da sua natureza ele consegue

reconhecer de forma eficiente se uma *query* de entrada pertence ou não aos comandos seguros esperados por uma aplicação. Nesse sentido, o AFD atua como um mecanismo de reconhecimento de padrões, testando cada caractere da query em tempo de execução para garantir que a sintaxe siga rigorosamente o fluxo permitido, evitando assim a injeção da cláusula UNION. O autômato criado para esta identificação é chamado de M1, ele também é definido formalmente pela quíntupla M1 = (Q, Σ, δ, q0, F) onde:

- M1 = ({q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11}, {a, b, c,..., z}, δ, q0, {q5})
- Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11}
- Σ = {a, b, c, ..., z, (espaço)}
- δ = (estado atual, entrada) = próximo estado
- q0 = estado inicial
- F = {q11}

O principal objetivo do autômato M1 é garantir que a sequência de entrada não contenha a palavra “union” seguida de “select”, que é um padrão malicioso de injeção SQL, reconhecendo assim um ataque potencial. Para a simulação será considerado que as *strings* que o autômato terá acesso serão tratadas e virão em minúsculo como ocorre em sistemas reais. O Autômato M1, incorpora regras de transição que garantem o reconhecimento preciso da entrada maliciosa em qualquer parte da *query* de entrada. Portanto as transições especiais do modelo M1 são definidas da seguinte forma:

- Regra de Início: O autômato fica em q0 (estado inicial) até que a entrada seja “u”.
- Regra de Recomeço: Se ocorrer qualquer outro caractere que não seja o esperado ou “u” ou “espaço” o autômato volta para q0.

- Regra da Ação da Letra “u”: de q1 a q4 se o próximo caractere for “u” o modelo retorna a q1 buscando a palavra “union” a partir do estado q1.

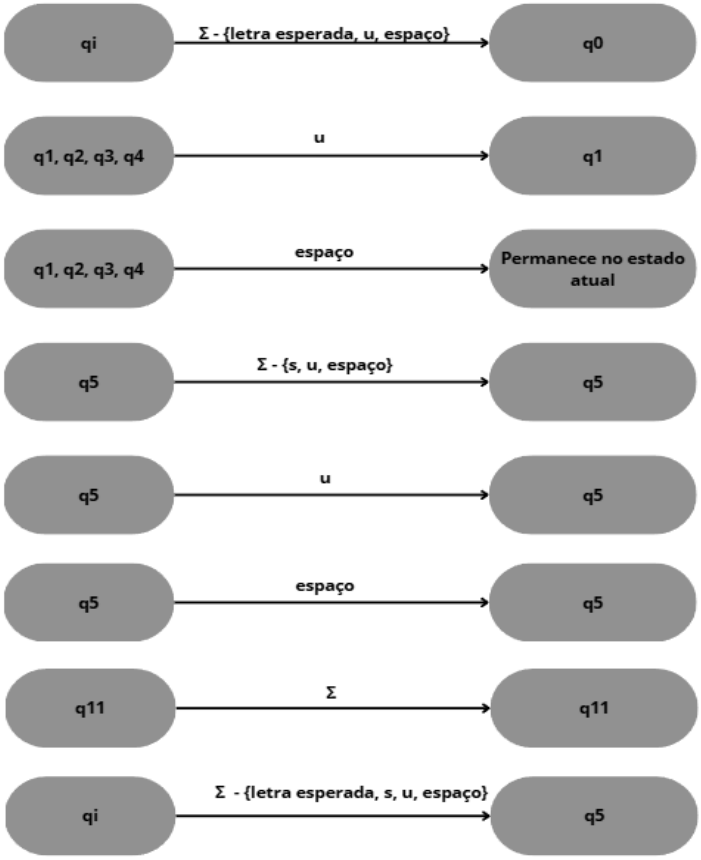
- Regra de Ignorar Espaço: de q1 a q4, a leitura de um “espaço” mantém o modelo no estado atual, permitindo espaços entre as letras da palavra, ele só avançará quando a letra correta for lida
- Regra de Espera: O autômato permanece em q5 ignorando a maioria dos caracteres, esperando a letra inicial “s” de "select", um “u” ou “espaço” priorizando a busca por select.
- Regra da Falha: Se o caractere esperado falhar, não for a letra correta, “s”, “u” ou “espaço”, o autômato retorna a q5. Garantindo que a máquina possa recomeçar a buscar “s” de "select" imediatamente.
- Regra do Estado Final: Ao alcançar q11 (estado final) qualquer caractere lido a partir de q11 mantém o autômato em q11. Uma vez que a sequência maliciosa já foi confirmada.

Todas estas regras fazem com que o autômato M1 busque a string maliciosa de forma que ele tolere erros na parte final da busca, mesmo que a busca pela palavra “select” falhe no meio, a máquina permanece em q5 se a palavra “union” já estiver sida lida. Deste modo o modelo M1 aceita a linguagem (L) de todas as strings que contêm a subcadeia “union” seguida em algum ponto por “select”.

- $L = \{x \in \Sigma \mid x = y \cdot \text{”union”} \cdot z \cdot \text{“select”} \cdot w \text{ para } y, z, w \in \Sigma \}$

Para consolidar a compreensão do comportamento do Autômato M1, essencial para a validação formal da sua proposta, a seguir são apresentados dois componentes cruciais. O resumo das regras de transição específicas e a tabela de transição. O resumo das regras oferece uma visão resumida das ações específicas que diferenciam o modelo M1 de um modelo de reconhecimento de subcadeia padrão. Já a tabela de transição é a formalização das transições, mapeando o autômato para cada estado e cada símbolo presente no alfabeto de entrada.

Figura 5 - Resumo das Regras de Funcionamento do Autômato M1



Fonte: Os autores (2025).

Tabela 4 - Tabela de Transição do Autômato M1

Estado Atual	**Transições Chaves**	**Ações de Espera/Recuperação**
q0	“u” → q1	Σ - {u, espaço} → q0; espaço → q0
q1	“n” → q2	“u” → q1; Σ - {n, u, espaço} → q0; espaço → q1

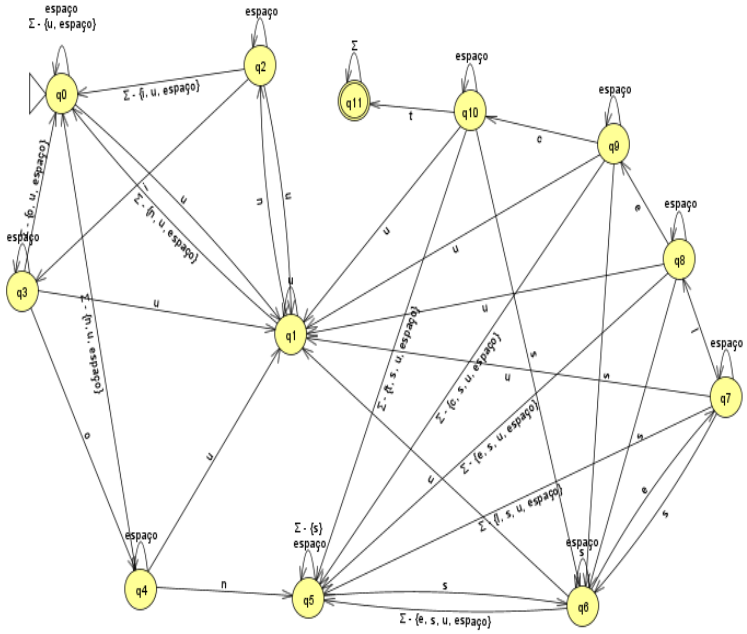
q2	“i” → q3	“u” → q1; Σ - {i, u, espaço} → q0; espaço → q2
q3	“o” → q4	“u” → q1; Σ - {o, u, espaço}→ q0; espaço → q3
q4	“n” → q5	“u” → q1; Σ - {n, u, espaço} → q0; espaço → q4
q5	“s” → q6	“u” → q5; Σ - {s, u, espaço} → q0; espaço → q5
q6	“e” → q7	“s” → q6; Σ - {e, s, u, espaço} → q5; espaço → q6; “u” → q1
q7	“l” → q8	“s” → q6; Σ - {l, s, u, espaço} → q5; espaço → q7; “u” → q1
q8	“e” → q9	“s” → q6; Σ - {e, s, u, espaço} → q5; espaço → q8; “u” → q1
q9	“c” → q10	“s” → q6; Σ - {c, s, u, espaço} → q5; espaço → q9; “u” → q1
q10	“t” → q11	“s” → q6; Σ - {t, s, u, espaço} → q5; espaço

		→ q10; “u” → q1
q11	“Σ” → q11	Aceita tudo que vier depois pois a entrada maliciosa já foi encontrada

Fonte: Os autores (2025).

Onde o símbolo Σ representa o alfabeto usado pelo modelo e a notação Σ - {letra} representa o alfabeto usado menos o caractere ou o conjunto de caracteres presentes na chave. Como resultado a seguir, é apresentada a imagem da modelagem compactada do autômato M1. Esta versão prioriza a legibilidade e a eficiência visual, usando símbolos para representar a volta ao estado inicial e o loop no estado final.

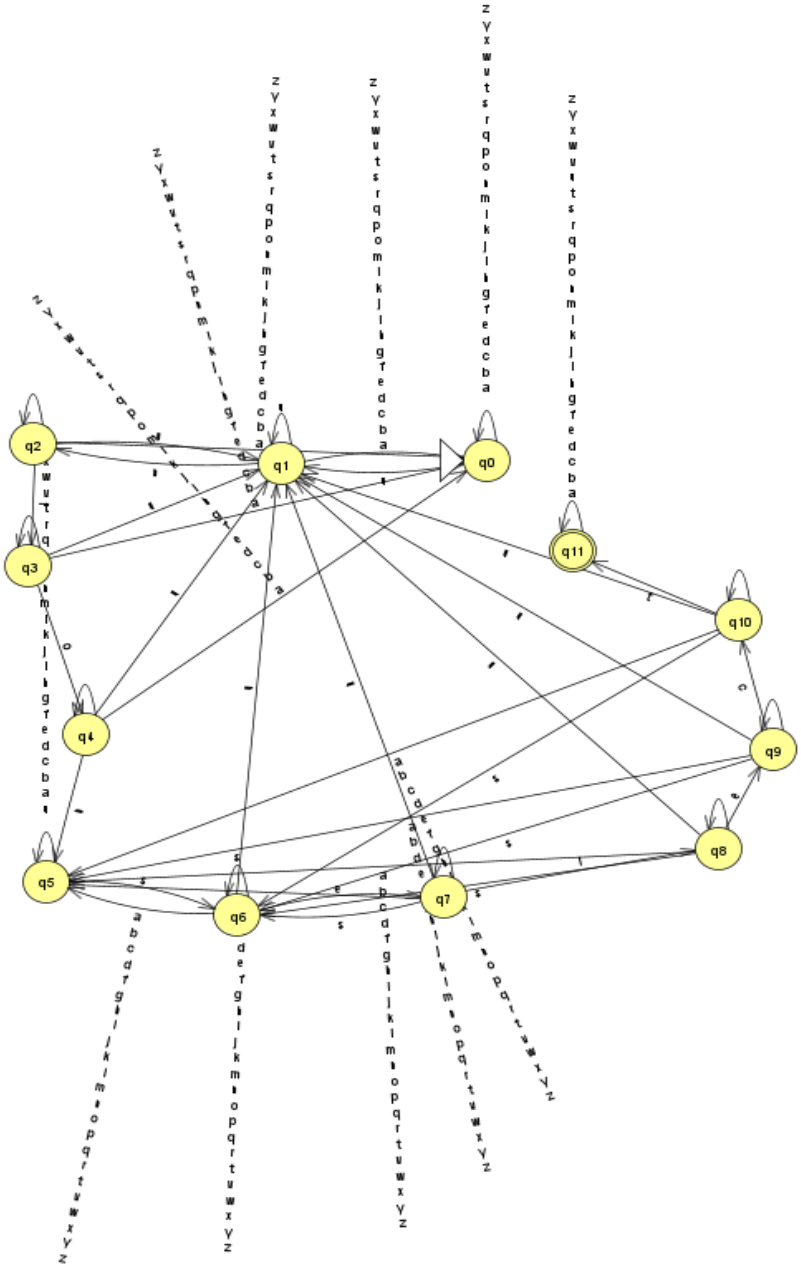
Figura 6 - Modelagem Compactada para Melhor Visualização do Autômato M1



Fonte: Os autores (2025).

A seguir é mostrada a imagem da modelagem completa e original do autômato M. Mostrando cada transição para cada símbolo do alfabeto e regras de transições adotadas.

Figura 7 - Modelagem Original do Autômato M1



Fonte: Os autores (2025).

A validação do autômato M1 foi realizada através de testes de reconhecimento em um conjunto de seis strings de entrada, com 3 delas contendo a palavra “union” seguida de “select” como e 3 delas não. O objetivo foi confirmar se as entradas com a subpalavra “union” seguidas de “select” tratadas como maliciosas e as que não tivessem não fossem tratadas como maliciosas.

Tabela 5 - Tabela de Resultados do Autômato M1

String	**Tipo**	**Identificada como Maliciosa**
select a from tabela union select c from tabela	Maliciosa	Sim
nomedoprojeto	Não Maliciosa	Não
select nome union select senha from contas	Maliciosa	Sim
nomedousuario	Não Maliciosa	Não
select cartao from cartoes union select senha from senhas	Maliciosa	Sim
nomedaconta	Não Maliciosa	Não

Fonte: Os autores (2025).

Os resultados obtidos nos testes de reconhecimento confirmaram integralmente o comportamento esperado do modelo M1. Já que o autômato foi capaz de identificar com sucesso as strings consideradas maliciosas, ou seja, ele conseguiu identificar as strings que continham a subcadeia “union” seguida por “select”, provando ser um mecanismo de identificação de ataques *UNION*

SELECT em queries de banco de dados SQL. A precisão na identificação destas strings confirma que o modelo atende ao objetivo proposto de ser um componente útil na prevenção e mitigação de ataques de injeção SQL.

4.3 Modelagem do Autômato para Identificação de Tautologia

Para identificar ataques de injeção NoSQL baseados em tautologia, optou-se por focar na detecção de operadores-chave de comparação que, quando empregados em campos de entrada de forma não autorizada, induzem a condição de *login* ou filtro a ser sempre verdadeira (tautológica). Um dos padrões mais comuns em injeções NoSQL é o uso do operador “$ne” (Not Equal, ou "diferente de") em expressões como {$ne: null}, que sempre retornam, se o campo existir, o valor lógico TRUE.

Desenvolveu-se um Autômato Finito Determinístico (AFD) com o objetivo de reconhecer a subpalavra “$ne” em qualquer posição das *strings* de entrada, modelando um autômato, denominado M2, capaz de detectar esse padrão específico de ataque. Definido formalmente pela quíntupla M2 = (Q, Σ, δ, q0, F) onde:

- Q - É um conjunto finito de estados possíveis não vazios do autômato;
- Σ - É um Alfabeto de símbolos de entrada;
- δ - Função de Transição
- q0 - É o Estado inicial, tal que q0 ∈ Q.
- F - É o conjunto de Estados Finais ou Estados de Aceitação, tal que F ⊆ Q.

O objetivo do autômato M2 é assegurar que a sequência de entrada não contenha a subcadeia “$ne”, indicando, assim, um potencial ataque de tautologia NoSQL.

Para a simulação, M2 é definido como:

- M2 = ({q0, q1, q2, q3} Σ, δ, q0, {q3})
- Q = {q0, q1, q2, q3}

- Σ = O conjunto de todos os caracteres possíveis (letras, números e símbolos especiais) que constituem uma *query* NoSQL, com foco nos tokens de transição: $, n e e.
- q0 = estado inicial
- F = {q3}

O Autômato M2 incorpora regras de transição que garantem o reconhecimento preciso da subpalavra “$ne” em qualquer parte da *query* de entrada. Portanto, as transições especiais do modelo M2 são definidas da seguinte forma:

- Ação do símbolo $ (Início do Padrão): Para todos os estados qi, onde i < 3 (de q0 a q2), a leitura do símbolo $ força o autômato a recomeçar o reconhecimento da sequência “ne”, levando-o diretamente ao estado q1 (prefixo $). Se, por exemplo, o autômato estiver no estado q2 e a próxima entrada for $, o autômato retorna para o estado q1.
- Ação de Retorno ao Estado Inicial (q0): Qualquer caractere do alfabeto Σ que não seja o esperado para dar continuidade à sequência “$ne” e que não seja o símbolo $, leva o autômato de volta ao estado q0 (estado inicial). Essa regra garante que o modelo permaneça no estado q0 até encontrar o primeiro caractere da palavra-chave ($).
- Regra do Estado Final (q3): Uma vez que o autômato alcance o estado q3 (estado final), onde a subpalavra “$ne” foi reconhecida, qualquer caractere lido em seguida mantém o modelo no estado final q3. Isso faz com que o ataque seja reconhecido independentemente dos caracteres que o precedem ou o sucedem na *query*.

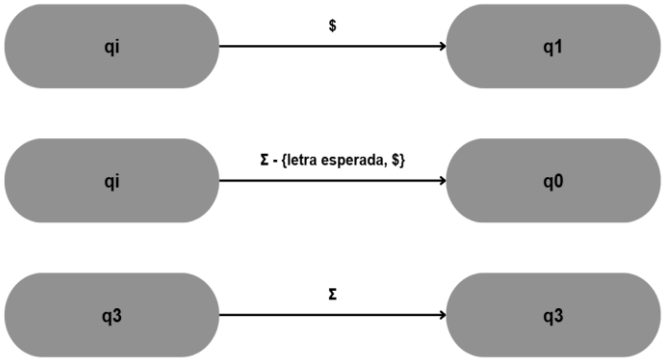
Dessa forma, a linguagem reconhecida, denominada L2, pelo autômato M2 é o conjunto de todas as palavras que possuem a subpalavra “$ne” em qualquer posição:

- $L2 = \{x \in \Sigma^* \mid \exists\ u,v \in \Sigma^* \text{ tais que } x = u \cdot \$ne \cdot v\}$

Para consolidar a compreensão do Autômato M2, crucial para a validação formal da detecção de tautologia NoSQL, apresentam-se dois componentes essenciais: o resumo das regras e a Tabela de Transição. O resumo das regras

oferece uma visão concisa das ações específicas que diferenciam o modelo M2 de um reconhecimento de subcadeia padrão, com foco na lógica de reinício baseada no símbolo $. A Tabela de Transição, por sua vez, formaliza completamente as transições δ, mapeando o comportamento do autômato para cada estado e símbolo do alfabeto de entrada Σ.

Figura 8 - Resumo das Regras de Funcionamento do Autômato M2



Fonte: Os autores (2025).

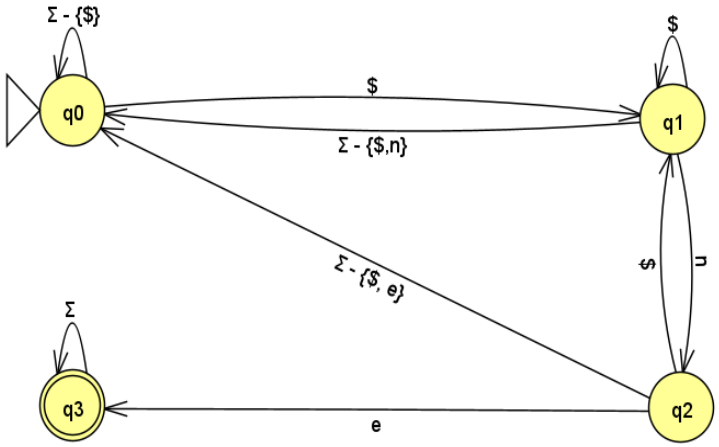
Tabela 6 - Tabela de Transição do Autômato M2

Estado Atual	Entrada	Próximo Estado
q0	$	q1
q0	Σ - {$}	q0
q1	n	q2
q1	$	q1
q1	Σ - {$,n}	q0
q2	e	q3
q2	$	q1
q2	Σ - {$, e}	q0
q3	Σ	q3

Fonte: Os autores (2025).

Onde o símbolo Σ representa o alfabeto usado pelo modelo e a notação Σ - {caracteres} representa o alfabeto usado menos o caractere ou o conjunto de caracteres presentes na chave. Como resultado, a seguir é apresentada a imagem da modelagem compactada do autômato M2. Esta versão prioriza a legibilidade e a eficiência visual, usando símbolos para representar a volta ao estado inicial e o *loop* no estado final.

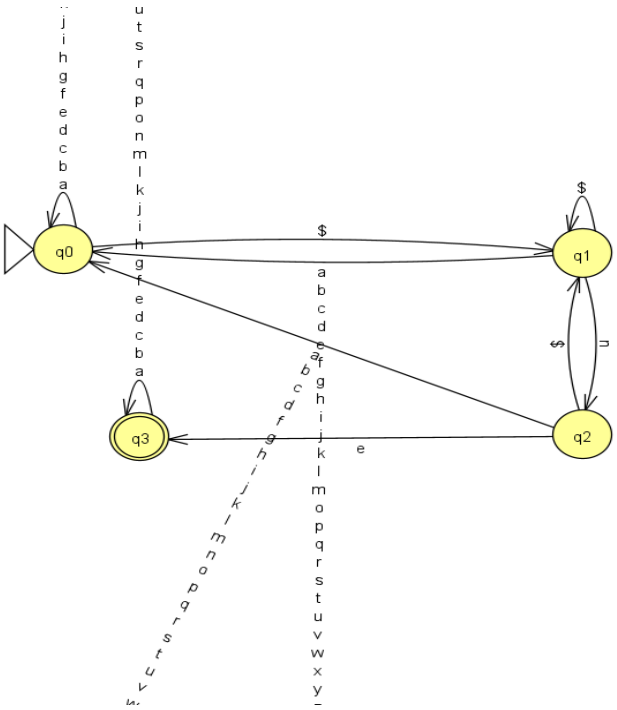
Figura 9 - Modelagem Compactada para Melhor Visualização do Autômato M2



Fonte: Os autores (2025).

A seguir é mostrada a imagem da modelagem completa e original do autômato M2. Esta modelagem apresenta cada transição para cada símbolo do alfabeto Σ e as regras de transições adotadas. Ela detalha a função δ na sua totalidade, expandindo a notação compacta “Σ - {...}” para ilustrar o caminho de cada símbolo no reconhecimento da subcadeia “$ne” e no tratamento de caracteres não relevantes.

Figura 10 - Modelagem Original do Autômato M2



Fonte: Os autores (2025).

A validação foi feita através de testes de reconhecimento em um conjunto de seis *strings* de entrada, com 4 delas contendo o operador “$ne” (Not Equal) de injeção NoSQL, e 2 delas não. O objetivo foi confirmar se as entradas com a subpalavra “$ne” eram tratadas como maliciosas e as que não continham a subpalavra não fossem tratadas como maliciosas.

Tabela 7 - Tabela de Resultados do Autômato M2

String	**Tipo**	**Identificada como Maliciosa**
userpassword =$ne	Maliciosa	Sim
username:tester123	Não Maliciosa	Não
project:name; $ne:null	Maliciosa	Sim
query:$ne:1or $gt:	Maliciosa	Sim
name:teste$ne	Maliciosa	Sim
user_name_01find:{name:jone}	Não Maliciosa	Não

Fonte: Os autores (2025).

Os resultados obtidos nos testes de reconhecimento confirmaram integralmente o comportamento esperado do modelo M2. O autômato foi capaz de identificar com sucesso as *strings* consideradas maliciosas, ou seja, ele conseguiu identificar as *strings* que continham a subcadeia “$ne”, provando ser um mecanismo eficaz de identificação de ataques de Injeção de Tautologia NoSQL para esse caso.

5. CONCLUSÃO

A crescente evolução dos ataques de injeção em bancos de dados SQL e NoSQL exige soluções de segurança com fundamentos em bases teóricas sólidas e com capacidade de operação eficiente em tempo real. Este trabalho, portanto, demonstrou que a aplicação de conceitos da Teoria da Computação,

especificamente autômatos finitos e linguagens regulares, permite a construção de uma abordagem viável e promissora para a detecção e prevenção de padrões maliciosos em consultas a bancos de dados, resolvendo, com isso, o problema citado anteriormente.

Ao analisarmos o que foi desenvolvido neste trabalho, podemos observar que os três autômatos finitos determinísticos propostos, M, M1 e M2, apresentaram um resultado satisfatório em seus respectivos cenários de aplicação. Inicialmente, o autômato M, projetado para reconhecer a subpalavra “union”, validou a capacidade de identificação desse padrão em qualquer posição de uma string contínua. Já o modelo M1, com maior complexidade, demonstrou eficácia na detecção de ataques UNION SELECT completos, reconhecendo a sequência maliciosa “union” seguida por “select” independentemente de espaçamentos ou outros tipos de caracteres. Por fim, o autômato M2 comprovou sua aplicabilidade na identificação de ataques de tautologia em sistemas NoSQL através do reconhecimento do operador “$ne”, que é um dos vetores mais comuns de injeção no contexto apresentado.

As simulações realizadas no JFLAP confirmaram que os autômatos propostos conseguem distinguir corretamente entre consultas legítimas e maliciosas nos cenários testados, demonstrando alta precisão nos conjuntos de validação. Esta eficácia demonstra que a modelagem formal baseada em teoria dos autômatos oferece garantias matemáticas de correção que abordagens heurísticas não podem proporcionar. A determinação dos modelos garante tempo de resposta linear em relação ao tamanho da entrada, característica essencial para implementação em sistemas de alta performance.

A principal contribuição deste trabalho está na utilização de uma metodologia formal para a construção de mecanismos de validação de entrada baseados em autômatos finitos, aplicável tanto a bancos SQL quanto NoSQL. Enquanto soluções tradicionais, como *prepared statements* e higienização de entrada, operam principalmente no nível de implementação, a abordagem proposta oferece uma camada adicional de proteção fundamentada em princípios matemáticos robustos. Além disso, a natureza composicional dos autômatos

permite que múltiplos padrões de ataque sejam modelados de forma modular e integrados em um sistema de detecção mais abrangente, para que seja possível mitigar uma gama mais ampla de vetores de ataque.

Entretanto, é importante ressaltar que houve limitações no estudo. Dentre elas, a utilização do JFLAP: embora adequado para prototipagem e validação conceitual, a ferramenta apresenta restrições quando se considera a escalabilidade para ambientes de produção com milhões de requisições simultâneas. Além disso, a efetividade dos autômatos também depende da qualidade do pré-processamento das entradas, como normalização e tratamento de maiúsculas/minúsculas.

Como continuidade deste estudo, recomenda-se a ampliação dos modelos formais desenvolvidos, incorporando novos padrões de ataque tanto em SQL quanto em NoSQL. As investigações futuras também podem explorar o uso de múltiplos autômatos funcionando em conjunto, através de arquiteturas paralelas ou hierárquicas, onde autômatos simples funcionariam como filtros iniciais rápidos e autômatos mais complexos analisariam em profundidade consultas suspeitas, melhorando a detecção sem perder desempenho. Por fim, propõe-se avaliar a implementação dos modelos em ambientes de escala e produção, substituindo o JFLAP por frameworks otimizados, a fim de validar o impacto real da abordagem em sistemas com cargas intensas e garantir sua viabilidade em ambientes corporativos.

Em suma, este artigo demonstrou como os autômatos finitos e linguagens regulares constituem ferramentas teóricas com alto poder para modelagem de mecanismos de segurança em ambos os paradigmas de banco de dados, oferecendo verificabilidade formal, eficiência computacional e modularidade. Portanto, a aplicação bem-sucedida dos modelos M, M1 e M2 na detecção de injeções UNION e tautologias NoSQL evidencia o potencial desta abordagem para fortalecer a segurança de sistemas de informação atuais, contribuindo para a construção de aplicações mais robustas e resilientes.

REFERÊNCIAS

ADITHI, G. S.; ADIGA, A.; PAVITHRA, K.; VASISHT, P. P.; KUMAR, V. *Secure, Offline Feedback to Convey Instructor Intent.* Proceedings of the IEEE Seventh International Conference on Technology for Education (T4E), Warangal, India, p. 105–108, 2015. DOI: 10.1109/T4E.2015.11

AWAD, Kassem Ubinski; CARDOSO, Luciano Santos; BUSSADOR, Alessandra. *NoSQL e segurança: um estudo de análise para prevenção de injeção em bancos de dados NoSQL.* Centro Universitário Dinâmica das Cataratas, Foz do Iguaçu, 2024.

BREVE, F. *Autômatos – Teoria da Computação.* 2009. Disponível em: https://www.fabriciobreve.com/material/tc/Texto1.pdf. Acesso em: 9 dez. 2025.

COLEN, B. *Dez vulnerabilidades cibernéticas frequentemente exploradas em 2024.* Microhard, 2024. Disponível em: https://microhard.com.br/dez-vulnerabilidades-ciberneticas-frequentes-2024/. Acesso em: 9 dez. 2025.

DETECTING and defeating SQL injection attacks. *International Journal of Information and Electronics Engineering*, jan. 2011. DOI: 10.7763/IJIEE.2011.V1.6.

FROZZA, Angelo Augusto; SCHREINER, Geomar André; MELLO, Ronaldo dos Santos. *Projeto de bancos de dados NoSQL.* In: *Short Courses of the 37th Brazilian Symposium on Databases (SBBD 2022).* Búzios, RJ, 2022. p. 26–52.

INOVTI. *Os 6 ciberataques mais frequentes em 2023 e como se preparar para 2024.* InovTI Blog, 2023. Disponível em: https://inovti.com.br/blog/os-6-ciberataques-mais-frequentes-em-2023-e-como-se-preparar-para-2024/. Acesso em: 9 dez. 2025.

JATANA, Nishtha; PURI, Sahil; AHUJA, Mehak; KATHURIA, Ishita; GOSAIN, Dishant. *A survey and comparison of relational and non-relational database.* International Journal of Engineering Research & Technology (IJERT), v. 1, n. 6, 2012. ISSN 2278-0181.

KHAN, W.; HUSSAIN, S.; AMIN, S.; RAZA, M.; AHMAD, N. *SQL and NoSQL database software architecture performance analysis and assessments—A systematic literature review.* Big Data and Cognitive Computing, v. 7, n. 2, art. 97, 2023. DOI: https://doi.org/10.3390/bdcc7020097.

KINJO, Paulo Renato. *Implementando autômatos finitos em Java.* Curso de Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2012.

LAGES, Gustavo P.; PEREIRA, Rafael T. *Estudo comparativo entre técnicas de detecção e prevenção de ataques de injeção SQL.* In: Congresso de Iniciação

Científica da Universidade Federal de Santa Maria (UFSM), Santa Maria, 2021. p. 1–6.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. *O modelo relacional.* Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2005. Produzido como parte da avaliação do curso MO-410: Introdução a Banco de Dados.

MONIRUZZAMAN, A. B. M.; HOSSAIN, Syed Akhter. *NoSQL database: new era of databases for big data analytics – classification, characteristics and comparison.* International Journal of Database Theory and Application, v. 6, n. 4, p. 1–14, 2013.

SILVA, P. S. S.; SANTOS, E. A. B. *Python para análise de dados em segurança cibernética utilizando regex.* Revista de Engenharia e Pesquisa Aplicada, v. 10, n. 1, p. 33–42, 2025. DOI: https://doi.org/10.25286/repa.v10i1.2508.

SOUZA, Larissa V.; BOER, Marcelo T. *Estudo comparativo entre os modelos de banco de dados NoSQL e SQL.* Trabalho de Graduação – Fatec Jales, Jales, 2023.

SOUZA, M. S.; RIBEIRO, S. E. S. B.; PIMENTA, I. A.; ALMEIDA, Y. O.; CARDOSO, F. J.; GOMES, R. L. *Detecção inteligente de injeção de SQL integrando ambientes de nuvem e borda.* Manuscrito, Universidade Estadual do Ceará, 2025.

SUN, Y.; VALGENTI, V. C.; KIM, M. S. *NFA-based pattern matching for deep packet inspection.* Proceedings of the 20th International Conference on Computer Communications and Networks (ICCCN), Lahaina, HI, 2011. p. 1–6. DOI: 10.1109/ICCCN.2011.6006095.