Sistema centralizado ledger para auditoria e identificação de dados adulterados em bancos de dados

Lucas Heilbuth Nazareth De Sousa[1]*; Daniele Caroline Lima da Silva[2]

[1] Universidade de São Paulo. Estudante. Belo Horizonte, Minas Gerais, Brasil
[2] Universidade de São Paulo. Orientadora. São Paulo, São Paulo, Brasil
*autor correspondente: lucasheilbuth@yahoo.com.br

Sistema centralizado ledger para auditoria e identificação de dados adulterados em bancos de dados.

Resumo (ou Sumário Executivo)

Empresas de tecnologia necessitam conectar suas soluções ao público para o disponibilizar os serviços, e isto possibilita que muitos hackers realizem ataques externos aos seus bancos de dados. Ademais, colaboradores realizam ataques internos seja por ocultação de erros ou obtenção de vantagens, em ambos os ataques são comprometidos a integridade dos dados. Nesse contexto este trabalho propõe um banco de dados Ledger, que armazena todos os dados históricos e assina digitalmente em cada bloco, garantindo a integridade dos dados e permitindo identificação do usuário e da transação de adulteração. A metodologia consiste em adaptar o sistema de banco de dados Ledger da Azure, simplificando algumas tabelas e disponibilizando em servidores locais, já que a solução atual exige o uso de serviço em nuvem. Como resultado obteve-se um sistema Ledger com tempo de inserção e remoção de 0,042 e 0,049 segundos, respectivamente e o algoritmo de recuperação de dados adulterados cresce a uma taxa linear de 0,0071 segundos por bloco inserido. Estes resultados corroboram com o objetivo de garantir mais eficiência que um banco de dados blockchain tradicional.

Palavras-chave: Blockchain; Ledger; Cibersegurança; Adulteração; SQL

Introdução

Durante o período da Covid 19, houve um aumento expressivo de aplicações web para hospedar banco de dados [BD] empresariais. Com esse crescimento, a cibersegurança tornou-se essencial para combater ataques internos e externos nas organizações. Um exemplo de ataque externo ocorrido durante a pandemia foi a ação de hackers que expuseram mais de 8,5 bilhões de dados sensíveis (Jithin e Subramanian; 2022).

Um dos métodos mais arriscados de invasão é a injeção SQL (Jithin e Subramanian; 2022), que compromete os três principais pilares de cibersegurança: a confidenciabilidade, integridade e disponibilidade (Kini et al; 2019). Além dos ataques externos, cerca de 30% das empresas relataram ataques internos, e 55% a 60% desses casos envolveram colaboradores com privilégios de administrador (Pradesh et al., 2024).

O uso de Banco de Dados Blockchain [BDB] visa solucionar este problema por ser um sistema que garante imutabilidade, usa criptografia de ponta a ponta e é um sistema descentralizado de alta disponibilidade (Habib et al; 2023). No trabalho de Muzammal et al. (2018) implementou-se um BDB que mostrou ser robusto a ataques DoS e DDoS, porém apresentou um tempo de 10 minutos para confirmar uma transação, como inserção ou remoção de um dado. Logo o tempo de processamento além de tornar-se um desafio, tende a crescer exponencialmente a cada bloco minerado.

O Banco de Dados Ledger Centralizado [BDLC] surge como uma alternativa para reduzir o alto custo computacional do BDB. Essa solução preserva a arquitetura centralizada adotada pela maioria das grandes empresas, ao mesmo tempo em que incorpora técnicas de

cibersegurança para a identificação de falhas de integridade. Além disso, o BDLC apresenta um tempo de processamento significativamente menor do que o BDB, tornando-se uma opção mais eficiente para auditoria e controle de dados.

Neste cenário, para o presente trabalho foi desenvolvido um sistema BDLC para um banco de dados utilizado na indústria automobilística. De modo que, este setor foi escolhido por sua vulnerabilidade a ataques cibernéticos, pela alta quantidade de dados sensíveis gerados e pela necessidade de assegurar a integridade e segurança dos dados para atender normas de proteção. Além disso, o uso de banco de dados auditáveis garantem integridade dos dados para os stakeholders do setor.

Para isso a aplicação deste trabalho será utilizado BD de documentos de requisitos na indústria automotiva. O processo de documentação inicia-se definindo o novo projeto de carro com os recursos e suas capacidades, então o time de produtos define requisitos de alto nível descrevendo capacidades gerais do carro. Em seguida, este time define critérios de aceitação de entrega e de performance esperada do algoritmo para assegurar que o desenvolvimento do software segue as normas estabelecidas, especialmente no que tange à segurança e confiabilidade do produto final. O time de produto envia esse documento para o time de software que desenvolve a documentação de requisitos que traduzem linguagem de programação em linguagem humana para auxiliar os desenvolvedores na implementação e na verificação se o software desenvolvido respeita as regras de negócios acordadas.

Estes requisitos são categorizados em: requisitos funcionais, que descrevem a lógica das funções do algoritmo, requisitos de performance que definem velocidade de processamento, e requisitos de segurança que definem as regras de segurança que devem ser cumpridas visando o bem-estar do passageiro e o cumprimento da legislação. No nível de hardware há também diversas unidades computacionais [ECU] presente no carro, e essas se comunicam através de protocolos de comunicação veicular. Nos documentos de software para carros, cada requisito é alocado para a ECU responsável por executar o conjunto do software. Identificar adulterações na ECU é fundamental para a correta alocação entre software e hardware.

De modo que, “Caso de Uso” e “Cenário do Usuário” representam as necessidades e os casos de experiências do usuário com o carro e o “Diagrama de Sequência” representa a sequência lógica que os requisitos devem ser executados relacionados a um Caso de Uso. Proteger a integridade dos dados relacionados aos cenários do usuário e às sequências de uso garantem que os requisitos de experiência do usuário e funcionalidade do produto sejam atendidos sem risco de manipulação.

O intuito deste trabalho é descrever a técnica utilizada desde (i) Modelagem do Banco de Dados Relacional, (ii) definição do banco de dados utilizado, (iii) implementar BDSQL pelo

framework MySQL, (iv) implementar BDLC e (v) resultados comparativos entre os dois métodos. Este trabalho será organizado da seguinte forma: Capítulo 2 é dedicado a apresentação do banco de dados e o algoritmo computacional. O capítulo 3 apresenta os principais resultados e comparação dos dois sistemas implementados. As conclusões são apresentadas no capítulo 4.

O objetivo deste estudo é desenvolver e validar um sistema BDLC para servidores SQL locais, garantindo auditoria e detecção de adulterações em bancos de dados da indústria automobilística.

Metodologia ou Material e Métodos

A metodologia deste trabalho consistiu em apresentar a modelagem conceitual, lógica e física e amostragem do banco de dados original “Requisito Cobertura”. Em seguida elaborou-se a modelagem lógica do BDLC e o descreveu-se cada tabela Ledger para controle de registros e validação dos blocos hashes.

Modelo Conceitual, Lógico e Físico

A primeira etapa do desenvolvimento do sistema de banco de dados [SBD] consistiu na modelagem conceitual, lógica e física, com o objetivo de representar as relações entre as tabelas. Essa modelagem forneceu uma estrutura organizada para a definição dos dados, permitindo o planejamento adequado da segurança e da integridade dos registros no sistema BDLC.

No projeto de Engenharia de Dados, o primeiro passo para construir o banco de dados foi definir o modelo conceitual, lógico e físico. De modo que o modelo conceitual representa a interação entre as tabelas, o modelo lógico especifica as colunas em cada tabela e o modelo físico descreve a estrutura de dados definida no framework escolhido.

A Figura 1 representa o SBD divido em 3 tabelas, a primeira armazena os requisitos de software, a segunda tabela contém a ECU que executa o requisito, sendo a relação entre requisito e ECU N:1, ou seja, cada requisito necessita ser executado em uma única ECU, mas cada ECU deve executar diversos requisitos. E a última tabela armazena as necessidades/experiências do usuário, e a relação entre requisito e caso de uso é de N:N isto significa que um requisito pode representar vários casos de uso, assim como um "Caso de Uso” vai ser descrito em diversos requisitos.

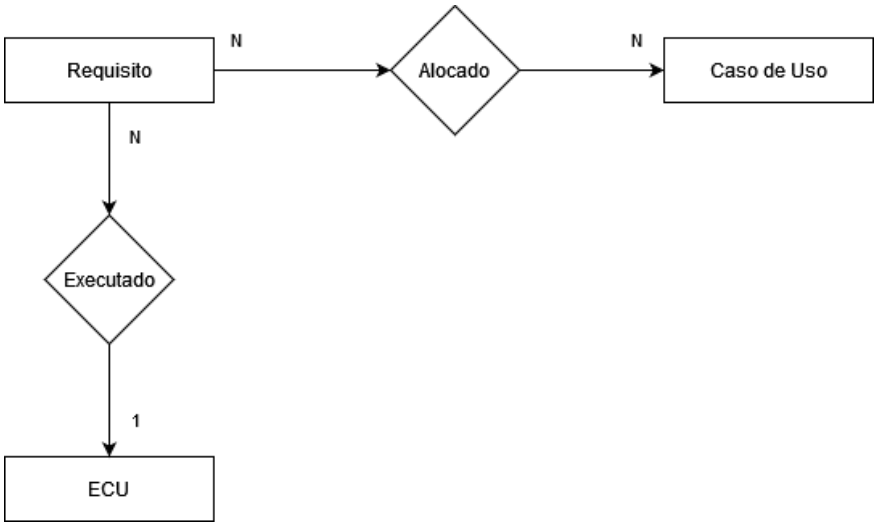

Figura 1: Modelo Conceitual
Fonte: Dados originais da pesquisa

A Figura 2 define o modelo lógico que fornece o detalhamento das colunas em cada tabela, onde a tabela “Requisito” contém o identificador de cada requisito, o conteúdo, o identificador do capítulo e o título. O título é utilizado para auxiliar a organização do documento de requisitos e com isso um conjunto de requisitos são adicionados dentro de um capítulo. Na tabela ECU a única informação relevante é a unidade computacional de cada requisito. E a tabela “Caso de Uso”, contém a coluna “Caso de Uso” e “Cenário do Usuário” que descrevem as necessidades do usuário, e o “Diagrama de Sequência” que contém o sequenciamento lógico dos requisitos.

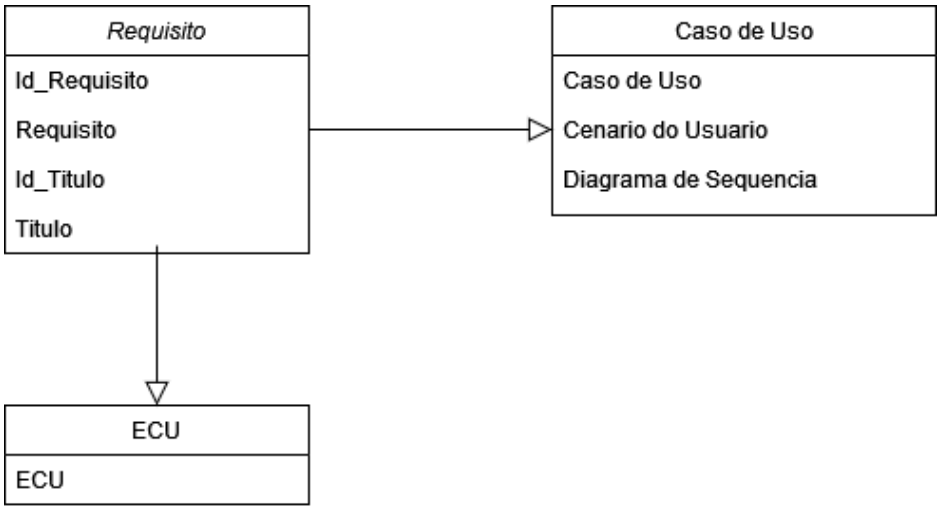

Figura 2: Modelo Lógico
Fonte: Dados originais da pesquisa

A Figura 3 apresenta o modelo físico, que especifica a estrutura do SBD desenvolvida no framework MySQL. Como as colunas armazenam dados textuais, optou-se pelo tipo

VARCHAR, com o limite de caracteres definido com base no maior registro identificado (pior cenário). No entanto, para a coluna "Requisito", devido ao grande volume de dados, foi necessário utilizar o tipo TEXT, pois seu tamanho excede a capacidade do VARCHAR. Além disso, nesse modelo, identificou-se a chave primária [PK], sendo o "Id_Requisito" o atributo responsável por relacionar os requisitos com as tabelas de ECU e Caso de Uso.

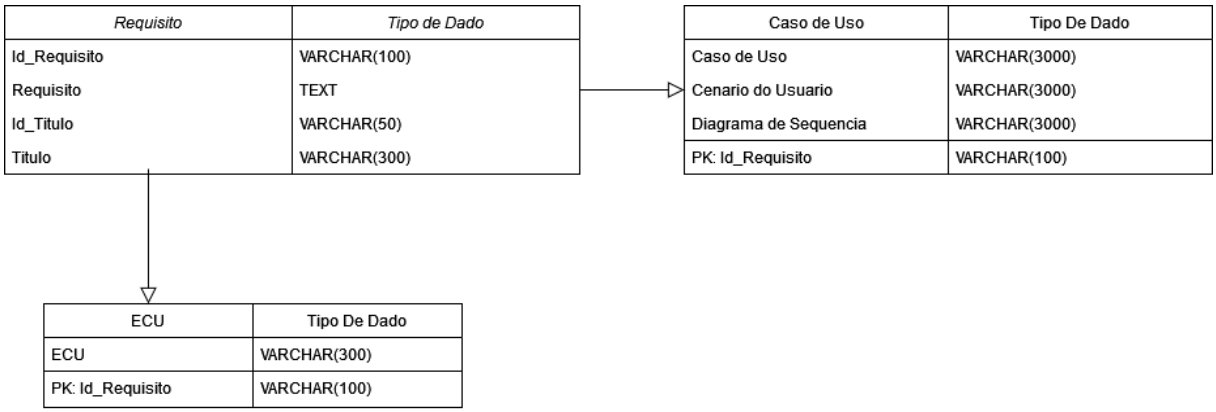

Figura 3: Modelo Físico
Fonte: Dados originais da pesquisa

Amostra do Banco de Dados

Em seguida, foi extraída uma amostra do banco de dados com o intuito de analisar seu conteúdo e compreender a relação entre as tabelas. Além disso, essa etapa permitiu validar o modelo desenvolvido no capítulo anterior. A primeira tabela de requisitos apresenta dados que demonstram como a leitura do estado das portas (aberta ou fechada) de um carro devem ser consideradas. Para evitar redundâncias, apenas o primeiro requisito de cada seção contém um título explícito, enquanto os requisitos subsequentes são automaticamente associados ao mesmo título do respectivo capítulo.

Tabela 1. Tabela de Requisitos

RequisitoId	Título	Requisito
ID_1	Status da Porta	
ID_2	ZCU CL Requisitos	
ID_3		O veículo deve ler o status da porta do motorista, do passageiro, traseira direita e traseira esquerda.
ID_4		O status da porta é binário definido como dois estados aberto e fechado, sendo o valor padrão fechado.

Fonte: Dados originais da pesquisa

Na tabela 2, foi exemplificado os dados que contém a coluna “PK Requisito Id” e a ECU, onde o nome da ECU utilizada (ZCU_CL) significa “Unidade de Controle da Zona” que são as ECUs centralizadas no carro que são responsáveis por comandar diversas ECUs periféricas. Este modelo de ECUs centralizadas é uma tendência cada vez mais presente na indústria automobilística.

Tabela 2. Tabela ECU

RequisitoId	ECU
ID_1	
ID_2	
ID_3	ZCU_CL
ID_4	ZCU_CL

Fonte: Dados originais da pesquisa

A tabela “Caso de Uso” é exibida na Tabela 3, nesta foi observado que a coluna “PK Requisito Id”, “Função”, escrita de acordo com a definição das funções de softwares. Em seguida, descreveu-se os dois documentos referentes ao “Diagrama de Sequência”, um quando o carro é trancado pela chave, e outro quando destranca. E para isso foi definido as experiências do usuário esperada pelo “Cenário do Usuário” e “Caso de Uso”.

Tabela 3. Tabela Caso de Uso

RequisitoId	Função	Diagrama de Sequência	Cenário do Usuário	Caso De Uso
ID_3	Gerenciar_Trancar_Destrancar	Trancar o veículo com key fob	Trancar/ destrancar o veículo com key fob ou proximidade	Trancar/ Destrancar o carro
ID_4	Gerenciar_Trancar_Destrancar	Trancar o veículo com key fob	Trancar/ destrancar o veículo com key fob ou proximidade	Trancar/ Destrancar o carro

Fonte: Dados originais da pesquisa

Visando simplificar o modelo e o desenvolvimento dos scripts, mesclou-se as 3 tabelas para representar todas as colunas relevantes esta tabela denominou-se “Cobertura de Requisitos”, como observa-se na Tabela 4.

Tabela 4. Tabela Cobertura Requisitos

RequisitoId	Título	Requisito	ECU	Função	Diagrama de Sequência	Cenário do Usuário	Caso De Uso
ID_1	Status da Porta						
ID_2	ZCU CL Requisitos						
ID_3		O veículo deve ler o status da porta do motorista, do passageiro, traseira direita e traseira esquerda	ZCU CL	Gerenciar_Trancar_Destrancar	Trancar o veículo com key fob	Trancar/destrancar o veículo com key fob ou proximidade	Trancar/ Destrancar o carro
ID_4		O status da porta é binário definido como dois estados aberto e fechado, sendo o valor padrão fechado.	ZCU CL	Gerenciar_Trancar_Destrancar	Trancar o veículo com key fob	Trancar/destrancar o veículo com key fob ou proximidade	Trancar/ Destrancar o carro

Fonte: Dados originais da pesquisa

Implementação do BDSQL

A implementação do banco de dados foi realizada no MySQL, utilizando o MySQL Workbench para gerenciamento e definição das tabelas. A escolha dessa tecnologia deve-se à sua robustez, suporte a transações ACID e ampla adoção na indústria. O script de auditoria e verificação de integridade foi desenvolvido em Python, utilizando a biblioteca MySQL Connector, devido à sua flexibilidade na execução de operações SQL e à compatibilidade com sistemas de auditoria de dados.

Após configurar o banco de dados inserindo os dados da planilha e o tipo de dados, foi realizado a consulta SQL pelo algoritmo Python, obtendo um resultado bem similar a Tabela 4 demonstrando que a conexão com o banco de dados e o script para consulta SQL está validado.

Implementação do BDLC

Para o desenvolvimento do modelo BDLC foi utilizado a documentação da Microsoft Azure como referência de features a serem implementadas, porém o código e a solução são fechados por isso optou-se por implementar o código em Python para fornecer uma solução BDLC para servidores locais. A arquitetura do sistema BDLC é demonstrado na Figura 4. Sendo que essa estrutura de BD adiciona novas colunas para guardar o histórico de alterações.

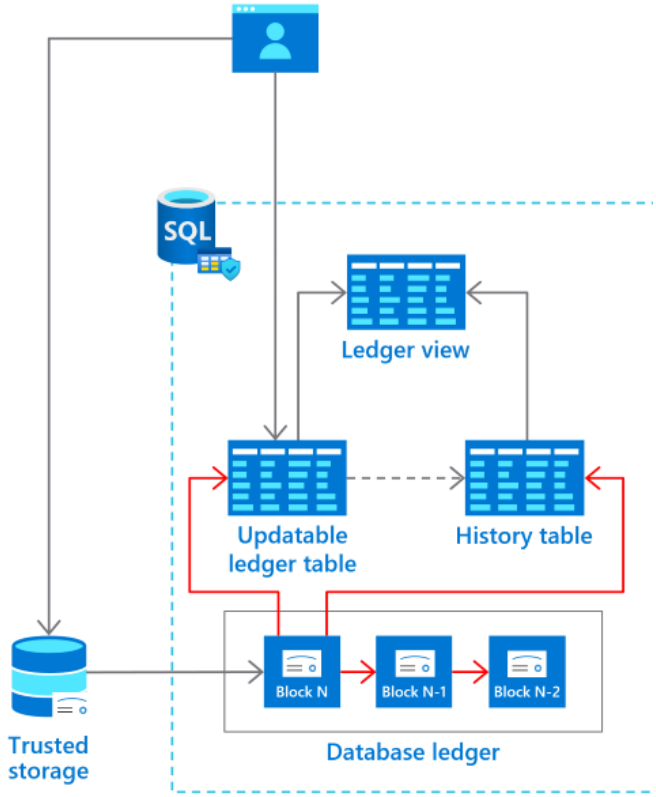

Figura 4: Modelo BDLC
Fonte: Azure Learn Ledger (2024)

O modelo lógico do sistema BDLC descreve o relacionamento das tabelas e o conteúdo das colunas. Na Figura 5 está exposto o modelo Original Azure, porém neste trabalho optou-se por realizar ajustes para simplificar o modelo. Primeiramente definiu-se que a Tabela Digest não será armazenada no banco de dados, pois é apenas uma combinação de tabelas previamente salvas, sendo a junção da tabela “Transações”, “Histórico Metadados” e “Operação Ledger”. Também foi adaptado a tabela “Histórico Hash”, permitindo que armazene além do Hash do bloco anterior, armazene o hash do bloco atual calculado, realizando um link entre o hash de cada bloco subsequente.

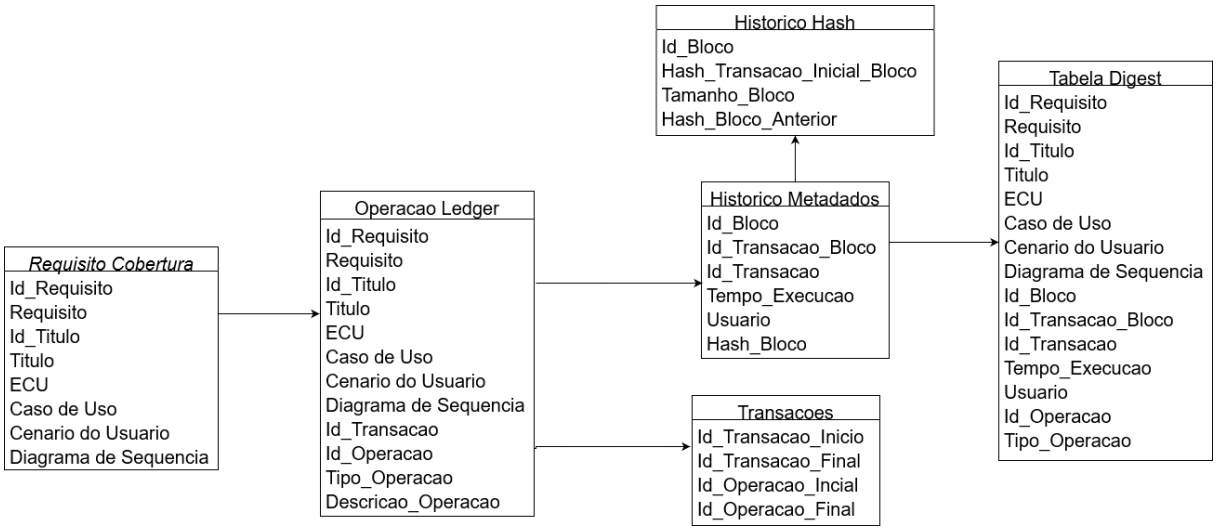

Figura 5: Modelo Lógico Original Azure
Fonte: Dados originais da pesquisa

A Figura 6 contém o modelo lógico adaptado neste trabalho em que cada bloco formado armazena um “Identificador” único e a partir disso calcula a assinatura digital do bloco, o valor hash.

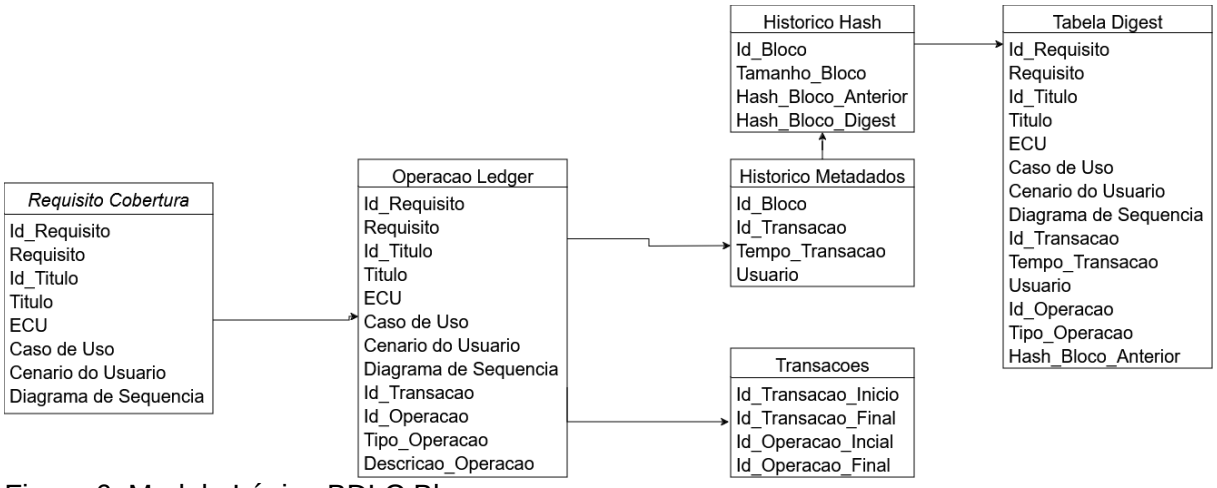

Figura 6: Modelo Lógico BDLC Bloco
Fonte: Dados originais da pesquisa

Tabela Transações

A tabela transações tem o objetivo de armazenar todas as transações realizadas na tabela, ou seja, gerar um histórico de alterações na tabela original. A transação é definida como um comando de alteração da tabela original podendo ser adulteração ou uma alteração legítima. Sendo que cada transação pode conter diversas operações, a operação é uma ação atômica SQL. Por exemplo a transação "Atualizar Tabela” é dividida entre inserir um novo dado e remover o antigo. Ou seja, com esta transação foi realizada duas operações.

A tabela transações, contém dados dos identificadores da transação e da operação. Se o comando for de remoção de dados, vai conter valores nas colunas “Final”, demonstrando a operação e transação em que o dado foi removido.

Blocos

No BDLC, o fechamento de blocos ocorre com base no número de transações ou no tempo decorrido. Blocos menores reduzem a perda de dados em caso de ataques, pois permitem a recuperação até o último estado confiável. No entanto, blocos menores também aumentam o custo computacional, especialmente durante a validação da auditoria Ledger. Dessa forma, a escolha do tamanho do bloco deve equilibrar segurança e desempenho computacional.

Operação Ledger

A tabela “Operação Ledger” associa cada operação ledger com o tipo de alteração realizada inserção, corresponde ou remoção, associa-se os valores 1 ou 2 para esses modos de alteração, respectivamente.

Tabela Histórico Metadados

A "Tabela Histórico Metadados” associa cada transação dentro do bloco com informações adicionais da execução como o autor e o tempo que o dado foi salvo no banco de dados. Estes dados são fundamentais para identificar a transação em que houve adulteração dos dados.

Tabela Digest

Para fins de verificação da integridade dos dados, adiciona-se a "Tabela Digest”, que é, uma tabela que contém todas as informações do bloco. Ou seja, concatena-se a tabela transações, histórico metadados e operação ledger. Esta tabela é utilizada para calcular o resultado da assinatura digital dos dados, o valor hash. E por ser uma composição das tabelas prévias não é guardada no banco de dados.

Tabela Histórico Hash

A tabela “Histórico Hash” é fundamental para o processo de audição pois é responsável por armazenar o código hash. O algoritmo SHA-256 foi utilizado para garantir a integridade dos dados armazenados no BDLC. Esse método gera um hash de 64 caracteres, com tamanho fixo, independentemente da entrada. Além disso, pequenas alterações nos dados resultam em hashes completamente diferentes, garantindo resistência contra adulterações. Por ser uma função criptográfica unidirecional, o SHA-256 impede a reconstrução da entrada a partir do hash gerado, tornando a falsificação dos dados extremamente difícil.

Este método SHA256 é utilizado para calcular Hash da “Tabela Digest” para cada bloco. Com isto, esta tabela relaciona o bloco com a assinatura digital dos dados, que é o valor hash.

Verificação da Audição Ledger

A audição Ledger garante integridade nos dados validando se houve dados adulterados na tabela histórica (Audição Ledger Azure; 2024). Este método valida a integridade dos dados históricos comparando com o valor hash. Para isso recalcula-se o “Hash” de cada bloco e compara-se ao valor armazenado na "Tabela Histórico Hash” caso haja incompatibilidade nos valores retorna falso e destaca-se a transação corrompida. Este método é oneroso computacionalmente e pode ser realizado simplificações de pular alguns blocos para reduzir o custo computacional (Audição Ledger Azure; 2024).

Este procedimento garante integridade dos dados, ou seja, a garantia que nenhum dado da tabela histórica foi alterado de forma maliciosa e que o histórico contempla todas as alterações já realizadas na tabela original. Para o procedimento ser efetivo deve-se reduzir a diferença do tempo do ataque e da detecção, por isso recomenda-se executar o método periodicamente. Também por ser um método computacional de custo elevado recomenda-se executar em momentos de baixa demanda do servidor (Audição Ledger Azure; 2024). Um exemplo de execução deste método é observado na Figura 7, o “Timestamp” é o momento da validação da tabela e “Result” retorna se o sistema BDLC sofreu um ataque de adulteração ou se o banco de dados é confiável.

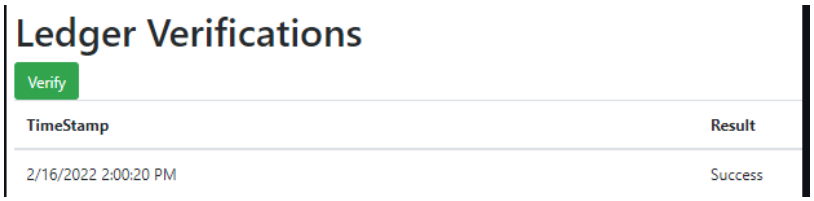

Figura 7: Verificação da Audição Ledger
Fonte: Azure Learn Ledger (2024)

Recuperação da Tabela Ledger após dados adulterados

Após executar a “Verificação da Audição Ledger” e retornar falso, o usuário pode executar o algoritmo de recuperação de tabela. Este método consiste em retornar o banco de dados para o último estado confiável do banco de dados. Para isso utiliza o identificador do bloco que foi corrompido no método de verificação da audição ledger, utiliza o valor antecessor ao identificador corrompido e retorna o banco para este estado. Neste contexto blocos grandes geram maiores perdas de dados e por isso a importância de conciliar o número de transações em cada blocos para evitar perdas de dados a cada adulteração, sem que haja aumento expressivo no tempo de execução do algoritmo.

Resultados Preliminares

Esta seção apresenta os resultados do trabalho, com o foco na apresentação das tabelas históricas gerada pelo algoritmo “Banco Dados Ledger”. Ademais, demonstra-se o algoritmo de detecção e recuperação de dados adulterados pela comparação da assinatura dos dados históricos (hash). Por fim, apresenta-se as métricas de performance e de segurança em detectar dados adulterados.

O script em Python permite as três principais operações CRUD: inserção, remoção e atualização. A tabela 5 demonstra as transações definidas para a tabela “Requisito Cobertura”.

Tabela 5. Transações na Tabela Original Requistos Cobertura

Id_Transação	Descrição
1	Inserir Requisito 1
2	Inserir Requisito 2
3	Inserir Requisito 3
4	Inserir Requisito 4
5	Remover ID_2
6	Atualizar ID_3 Novo Requisito 3
7	Atualizar ID_4 Atualizado Requisito 4

Fonte: Resultados originais da pesquisa

Para inserir o requisito, o único argumento é o texto do requisito enquanto para remover é necessário a identificação do requisito. E para executar a atualização solicita-se o identificador e o novo texto do requisito. Após a operação na tabela original "Requisitos cobertura", insere-se os dados históricos nas tabelas "Ledgers". Então, é iniciado com a tabela "Operação Ledger" que armazena o histórico de operações e o tipo de operação de cada transação. Por fim, como resultado têm-se a tabela 6.

Tabela 6. Tabela Operações Ledger

Id_Requisito	Requisito	Id_Transação	Id_Operação	Tipo_Operacao_Ledger
ID_1	Requisito 1	1	1	Inserir
ID_2	Requisito 2	2	1	Inserir
ID_3	Requisito 3	3	1	Inserir
ID_4	Requisito 4	4	1	Inserir
ID_2	Requisito 2	5	1	Remover
ID_3	Requisito 3	6	1	Remover
ID_3	Novo Requisito 3	6	2	Inserir
ID_4	Requisito 4	7	1	Remover
ID_4	Atualizado Requisito 4	7	2	Inserir

Fonte: Resultados originais da pesquisa

Conforme a tabela 6 observa-se que a transação 6 e 7 teve duas operações, pois cada transação representa uma consulta SQL e quando a consulta é de atualização é armazenado na tabela duas operações. Tais consistem na remoção do valor antigo e na inserção do valor novo. Através desta tabela, tem-se o histórico de todas as operações realizadas na tabela original em ordem cronológica o que auxilia na compreensão das atualizações e identificações de possíveis adulterações na tabela.

Em seguida o software gera a tabela de "Transações" que relaciona a transação de inserção do requisito e a transação que foi removida. A tabela 7 pode ser mesclada com outras tabelas com o intuito realizar uma análise histórica e identificar erros.

Tabela 7. Tabela Transações

Id_Transação_Inicio	Id_Transação_Final	Id_Operação_Inicio	Id_Operação_Final
2	5	1	1
3	6	1	1
4	7	1	1

Fonte: Resultados originais da pesquisa

As colunas "Id Transação Inicio" e "Id Operação Inicio" representam operações de inserção e as colunas "Id Transação Final" e "Id Operação Final" são operações de remoção,

com isso obtém-se todo o histórico desde a inserção até a remoção do requisito. Por ser uma tabela apenas de identificadores, recomenda-se mesclar com outras tabelas como histórico metadados para uma análise histórica mais detalhada.

Em seguida, gerou-se a tabela 8 que armazena dados pessoais de cada transação, como o usuário e o horário de cada transação. Esta tabela realiza a gestão de blocos, que de acordo com a definição do número de transações e tempo limite gera-se um novo bloco com assinatura digital hash. Para este trabalho foi definido um limite de duas transações por bloco ou 10 minutos.

Tabela 8. Tabela Histórico Metadados

Id_Bloco	Id_Transação	Horário_Transação	Usuário
1	1	2024-12-11 15:10:15	Lucas
1	2	2024-12-11 15:10:20	Lucas
2	3	2024-12-11 15:10:25	Lucas
2	4	2024-12-11 15:10:40	Lucas
3	5	2024-12-11 15:10:50	Carlos
3	6	2024-12-11 15:11:05	Bianca
4	7	2024-12-11 15:10:30	Bianca

Fonte: Resultados originais da pesquisa

Para simulação foi realizado transações em sequência, considerando um tempo mínimo entre cada transação, de forma que apenas o limite de transações seja necessário para criar novos blocos.

Em seguida foi implementado o algoritmo da tabela “Digest”, as colunas são demonstradas na Figura 6. Em que se mesclou a tabela original “Requisito Cobertura”, “Operação Ledger” e “Histórico Metadados”. A concepção dessa tabela é de armazenar todos os dados de cada operação para gerar a assinatura digital única do bloco. Além disso adicionou-se o hash do bloco anterior, de forma que o hash atual depende do hash anterior, criando um vínculo encadeado entre os blocos. Com isto, para que ocorra a alteração de um bloco hash antigo é necessário recalcular o hash de todos os blocos subsequentes dificultando consideravelmente a adulteração.

Por fim tem-se a última tabela, “Histórico Hash” armazena o código hash calculado da tabela “Digest”, e o Hash anterior, desta forma se houver ataque de adulteração de dados deve-se descobrir todos os hashes dos blocos subsequentes e alterar os valores do hash anterior, de forma que o hash digest atual seja igual ao próximo hash anterior como demonstrado na Tabela 9.

Tabela 9. Tabela Histórico Hash

Id_Bloco	Tamanho_Bloco	Hash_Anterior	Hash_Digest
1	2		2909f888ba...

2	2	2909f888ba9...	2cc3977a41....
3	2	2cc3977a41....	d512ae115e...
4	1	d512ae115e...	475e4673a0...

Fonte: Resultados originais da pesquisa

Identificação de erros e adulteração através de dados históricos

O uso do código hash garante ao usuário que todos os dados históricos não foram adulterados, e caso haja adulteração retorna-se o estado dos dados para o bloco anterior a adulteração. Com isto, pode-se utilizar com segurança os dados históricos para compreender o avanço da tabela e identificar possíveis erros e adulteração.

Na visualização dos registros de transações é recomendado mesclar as tabelas históricas de acordo com a análise definida. Um exemplo de mesclagem é demonstrado na tabela 10 no qual mesclou-se a tabela “Transações” e “Histórico Metadados”.

Tabela 10. Visualização dos dados históricos

Requisito	Id_Transacao_Inicio	Id_Transacao_Final	Usuario_Inicio	Usuario_Final
Requisito 2	2	5	Lucas	Carlos
Requisito 3	3	6	Lucas	Bianca
Requisito 4	4	7	Lucas	Bianca

Fonte: Resultados originais da pesquisa

De acordo com o conhecimento do negócio e os objetivos do documento do requisito infere se as operações de inserção e remoção são coerentes, e então identifica-se transações suspeitas que o usuário pode desejar remover e retornar a um estado anterior a transação suspeita.

Recuperação de dados Adulterados

Para o hacker e obter sucesso na alteração de dados históricos necessita iniciar uma remoção ou atualização nos bancos de dados históricos, encontrar o novo valor hash, por força bruta ou acesso a todos os bancos históricos, repetir o processo para todos os blocos subsequentes até o último e alterar todos os valores hashes na tabela Histórico Hash antes do sistema criar o próximo bloco definido em 10 minutos. O recalculo dos blocos sequentes torna-se necessário por utilizar o hash antigo, no cálculo do hash atual, tornando computacionalmente muito caro alterar blocos antigos. Dessa forma, quanto mais antigo o dado histórico mais seguro fica na rede blockchain.

Além disso, foi implementado uma feature que após identificar o bloco adulterado, recupera o banco de dados até o último estado confiável. Para isso, recomenda-se que o

usuário execute recorrentemente o algoritmo de recuperação de dados para limpar qualquer dado adulterado por ataque externo ou interno do banco de dados. Este algoritmo é fundamental para garantir segurança aos stakeholders que todos os dados históricos representam todas as modificações já feitas na tabela original, porém é caro computacionalmente, pois necessita recalcular todos as assinaturas hashes dos blocos e ainda modificar o SBD para o estado seguro.

Métricas de desempenho do Sistema BDLC

O objetivo deste trabalho é adicionar camadas de segurança em um banco de dados SQL, para isso torna-se fundamental calcular o tempo gasto para executar essas features adicionais. A tabela 11 expõe o tempo gasto em segundos para operações CRUD na tabela original, que é um tempo similar a uma aplicação apenas SQL em comparação com o tempo gasto para adições nas tabelas históricas Ledger. Foi utilizado uma média de 100 amostras em cada operação para cada tabela analisada visando reduzir as variações amostrais.

Tabela 11. Tempo médio de 100 amostras para operações CRUD nas tabelas Ledger

Operações	Tempo Tabela Requisito Cobertura	Tempo Tabela Histórica	Tempo Total	Tempo Total/Tempo Requisito Cobertura
Inserir	0,0033	0,0387	0,042	12,7
Remover	0,0027	0,0463	0,049	18,15
Atualizar	0,0008	0,0502	0,051	637,5

Fonte: Resultados originais da pesquisa

Observa-se pela inserção em 5 tabelas históricas que guardam as transações, metadados e o código hash, que houve um aumento considerável do tempo de execução ao utilizar banco de dados ledger. Principalmente quando se compara o tempo total com o tempo da operação na tabela original. A maior diferença foi observada na operação atualizar, pois a operação CRUD é apenas de alterar dados em um registro criado, enquanto na tabela histórica necessita-se criar 2 registros o primeiro para adicionar a operação remover e o segundo com a operação de inserir.

Outra camada de segurança que se tornou fundamental a análise de performance é a recuperação de dados até o estado seguro sem adulteração. Para isso foi executado diversos cenários de adulteração em distintos blocos para mensurar o tempo gasto e inferir padrões matemáticos de performance. Foi simulado adulteração de dados históricos entre o bloco 1 e o bloco 100, executando 10 vezes para cada bloco, com isto gerou-se a Figura 8.

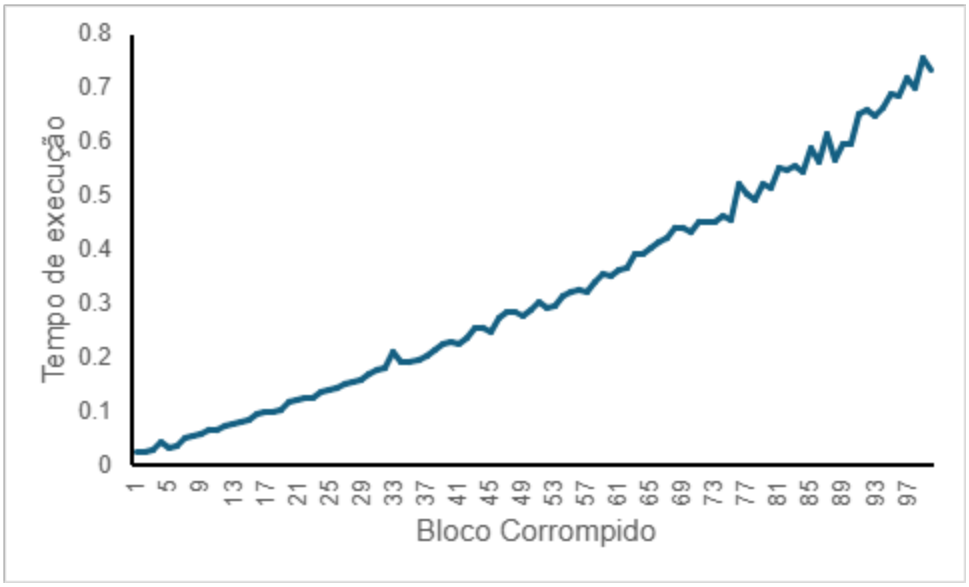

Figura 8. Tempo de execução em função do bloco corrompido
Fonte: Resultados originais da pesquisa

O algoritmo de recuperação de dados inicia do bloco 1, ao invés do mais recente, pois caso haja mais de 1 bloco corrompido deve-se retornar ao estado anterior ao primeiro bloco corrompido. Portanto para reduzir o custo computacional e simplificar o código foi selecionado essa abordagem. E com isto nota-se pelo gráfico um crescimento linear do tempo de execução em função do bloco corrompido.

Como o sistema BDLC acumula todos os registros históricos, é esperado que times de software enfrentem alto número de blocos minerados, e com isto os desenvolvedores podem sofrer impacto do tempo de execução para sistemas maiores. Portanto recomenda-se a constante medição e avaliação do tempo gasto, e quando se conclui que o tempo está alto, sugere-se algoritmo de limpeza de blocos, no qual define-se um limiar e mantém o BDLC apenas os últimos blocos de acordo com o limiar. O algoritmo de limpeza possui limitações, pois perde o histórico e, portanto, a recuperação de blocos adulterados que foi limpo da análise.

A partir do gráfico, foi inferido a equação matemática que relaciona o bloco corrompido com o tempo gasto, para permitir mensurar o tempo gasto de acordo com a escalabilidade do sistema. Para isso foi modelado uma função linear, pela similaridade do gráfico com esta função e definiu-se que no bloco 0, o tempo de execução é 0, pois não há dados de comparação de hash. Então gerou-se a eq. (1) que modela a escalabilidade do sistema.

$$Y = Y(0) + \frac{\Delta y}{\Delta x} X = 0 + \frac{(0{,}735 - 0{,}029)}{100 - 1} X = 0{,}00713\, X \qquad (1)$$

em que Y(0) representa o tempo gasto para o bloco 0, que foi definido como 0, Y o tempo gasto, X o número do bloco corrompido e delta x e delta y, representa a variação do número de bloco e a variação do tempo gasto, respectivamente.

Comparação dos atributos entre SQL, Blockchain e Ledger

Em seguida realizou-se a análise comparativa entre os 3 Sistemas de Banco de Dados descritos no trabalho para avaliar as vantagens e desvantagens de cada sistema e auxiliar os desenvolvedores na podenração do sistema adequado para implementar, com isto foi gerada a tabela comparativa 12.

Tabela 12. Características entre os Sistemas de Banco de Dados

Características	SQL	Blockchain	Ledger
Arquitetura	Centralizada	Descentralizada	Centralizada
Identificar Adulteração em Tabelas Históricas	Não	Sim	Sim
Método de criar novos blocos	Não possui	Sistema de votação descentralizada	Cálculo do hash centralizado
Tempo para criar novos blocos	Não possui	Alto	Baixa
Segurança de Integridade	Baixa	Alta	Alta
Performancen Operações CRUD	Alta	Baixa	Média
Tabelas históricas	Não	Sim	Sim
Local de armazenamento das tabelas históricas	Não possui	Todos nós que participam do sistema de votação	Apenas no banco de dados central

A primeira caracaterística de comparação é a arquitetura no qual SQL e Ledger são centralizados e o blockchain é descentralizado, já que depende de um sistema de votação no qual o bloco minerado deve receber a aprovação majoritária dos nós. Em questão da identificação de adulteração nas tabelas históricas o Blockchain e o Ledger possuem esta capacidade já que calculam uma assinatura digital hash para cada bloco permitindo detectar alterações que modificam o valor dessa assinatura.

O método de criar novos blocos também distingue os sistemas já que o blockchain depende do sistema de votação descrito e o ledger é um cálculo automático após atingir o limite do número de transações ou atingir o tempo limite, gerando-se uma assinatura para todos os dados da transação. E o tempo para criar novos blocos é alto no Blockchain pois o minerador necessita utilizar a força bruta para calcular um hash correspondente, que é uma

operação cara computacionalmente e os nós devem aceitar esse cálculo, enquanto que no Ledger o sistema calcula automaticamente o Hash de acordo com os dados armazenados.

A segurança da integridade é alta para o blockchain e ledger devido a identificação de adulteração e retorno para o estado confiável trazendo maior garantia que os dados armazenados não foram alterados inapropriadamente, e o SQL por não possui a arquitetura de blocos hashes, não possui essa garantia. Mas em questão de performance o SQL demonstra ser o melhor sistema com respostas rápidas as operações CRUD, enquanto ledger e blockchain demandam maior tempo de processamento.

O local de armazenamento das tabelas históricas do blockchain é descentralizado, armazena-se em todos os nós que votam no bloco minerado enquanto o ledger apenas guarda os dados no nó centralizado.

Conclusões ou Considerações Finais

Este trabalho atendeu aos objetivos especificados de implementação com base na documentação Azure um BDLC, para isso foi adaptado a estrutura das tabelas históricas Azure Ledger, para simplificar e reduzindo a complexidade e o tempo computacional da inserção de colunas que o autor considerou não obrigatórias. Como resultado de performance obteve-se um tempo médio de inserção/remoção de requisitos de 0,042 e 0,049 segundos, assim gasta-se 12,7 e 18,15 vezes mais tempo para realizar a operação em comparação com um banco de dados SQL. Porém o autor avalia que o ganho de segurança neste sistema deve ser ponderado para optar pelo uso deste sistema. Em questão de tempo gasto para recuperar blocos adulterados, notou-se um comportamento linear que pode ter um custo computacional alto para alto número de blocos. Para solucionar isto, o autor sugere futuros pesquisas que implementa um recorte de blocos, isto é, comparar o número de blocos com o limiar e remover os blocos antigos que ultrapassam esta comparação. Também outra pesquisa fundamental é a comparação do tempo gasto com blocos adulterados para diferentes métodos de hash, e comparar com o SHA256 que foi utilizado neste trabalho.

Referências

Habib, M. A.; Manik, M. M. H.; Zaman, S. 2023. A Blockchain-based Technique to Prevent Grade Tampering: A University Perspective. In: *International Conference on Electrical, Computer and Communication Engineering (ECCE), 2023,* Chittagong, Bangladesh, Anais... p. 1-6. Disponível em: <https://ieeexplore.ieee.org/document/10101502>. Acesso em 19 out. 2024.

Chowdhury, M. J. M.; Colman, A.; Kabir, M. A.; Han, J.; Sarda, P. 2018. Blockchain Versus Database: A Critical Analysis. In: *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big*

Data Science And Engineering (TrustCom/BigDataSE), 2018, New York, NY, USA. Anais... p. 1348-1353. Disponível em: < https://ieeexplore.ieee.org/abstract/document/8456055 >. Acesso em 19 out. 2024.

Muzammal M; Qu Q; Nasrulin B; Skovsgaard A. 2019. ChainSQL: A Blockchain Database Application Platform. In: *Chinese Academy of Sciences*, 2019, Beijing, China. Anais… p. 1-9. Disponível em: <https://arxiv.org/pdf/1808.05199v5>. Acesso em 19 out. 2024.

Microsoft Azure. 2024. Database Verification. Disponível em: <https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-database-verification?view=sql-server-ver16&preserve-view=true >. Acesso em 19 out. 2024.

Pradesh, G.; Sangeetha, D.; Kishore, V. R.; Sharan, L. S. 2024. Detection and Mitigation of Insider Attacks in Financial Systems. In: *International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2024, Chennai, India. Anais... p. 1-8. Disponível em: <https://ieeexplore.ieee.org/document/10602199>. Acesso em 19 out. 2024.

Jithin V. O.; Subramanian N. 2022. SECURE-D:Framework For Detecting and Preventing Attacks in SQL and NoSQL Databases. In: *10th International Symposium on Digital Forensics and Security (ISDFS)*, 2022, Istanbul, Turkey. Anais... p. 1-5. Disponível em: < https://ieeexplore.ieee.org/document/9800805>. Acesso em 19 out. 2024.

Kini, S.; Patil, A. P.; Pooja, M.; Balasubramanyam, A. 2022. SQL Injection Detection and Prevention using Aho-Corasick Pattern Matching Algorithm. In: *3rd International Conference for Emerging Technology (INCET), 2022, Belgaum, India. Anais... p. 1-6.* Disponível em: <https://ieeexplore.ieee.org/document/9825040>. Acesso em 19 out. 2024.

TBC Yang X.; Wang S.; Li F.; Zhang Y.; Yan W.; Gai F.; Yu B.; Feng L.; Gao Q.; Li Y. 2022. Ubiquitous Verification in Centralized Ledger Database. In: *IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, Kuala Lumpur, Malaysia. Anais... p. 1808-1821. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9835536>. Acesso em 19 out. 2024.