UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ÍTALO CAVALCANTE DE ABREU

PROCESSAMENTO DE CONSULTA ORIENTADO A PROPÓSITOS: UMA EXTENSÃO DA LINGUAGEM SQL

FORTALEZA

2021

ÍTALO CAVALCANTE DE ABREU

PROCESSAMENTO DE CONSULTA ORIENTADO A PROPÓSITOS: UMA EXTENSÃO DA LINGUAGEM SQL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Javam de Castro Machado

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A99p Abreu, Ítalo Cavalcante de.
Processamento de consulta orientado a propósitos: uma extensão da Linguagem SQL / Ítalo Cavalcante de Abreu. – 2021.
42 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2021.
Orientação: Prof. Dr. Javam de Castro Machado.

1. Sistemas de Gerenciamento de Banco de Dados. 2. Processamento de consulta. 3. SQL. 4. Propósito. 5. Proteção de dados. I. Título.

CDD 005

ÍTALO CAVALCANTE DE ABREU

PROCESSAMENTO DE CONSULTA ORIENTADO A PROPÓSITOS: UMA EXTENSÃO DA LINGUAGEM SQL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Javam de Castro Machado (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará (UFC)

Prof. Me. Paulo Roberto Pessoa Amora
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dr. Victor Aguiar Evangelista de Farias
Universidade Federal do Ceará (UFC)

Para os meus pais, que são meu exemplo e meu orgulho.

AGRADECIMENTOS

Aos meus pais, por investirem em mim e em meus estudos, por sempre me apoiarem em minhas decisões e por me darem exemplo de vida.

Aos meus irmãos que sempre estiveram ao meu lado, mesmo que muitas vezes distantes fisicamente.

A todos os professores que participaram de minha formação desde o ensino fundamental.

A Universidade Federal do Ceará e a todos os professores que contribuíram para minha formação profissional através de um ensino de qualidade inestimável. Em especial, ao meu orientador, Prof. Dr. Javam Machado que me deu a oportunidade participar do laboratório de pesquisa e que me isentiva a estar sempre estudando.

Ao Laboratório de Sistemas de Banco de Dados (LSBD) e aos membros da sua secretaria por oferecer uma estrutura e apoio espetacular para o desenvolvimento das pesquisas.

Aos colegas do LSBD, em especial ao doutorando Me. Daniel Praciano, meu supervisor e com quem trabalho no desenvolvimento de pesquisas, e aos colegas do Lab4 que sempre ajudam ao levantar discussões produtivas e apoiam uns aos outros.

Aos membros da banca, Prof. Dr. José Maria da Silva Monteiro Filho, Prof. Me. Paulo Amora e Prof. Dr. Victor de Farias por se disporem a avaliar esse trabalho e sugerir melhorias.

Agradeço também ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“ Todos encontram o que realmente procuram - mesmo sem saber o que estão procurando. ”

(C. S. Lewis)

RESUMO

Com a população global cada vez mais conectada e os serviços e processos informatizados, nunca houve tamanha disponibilidade de dados a serem processados, analisados e estudados, o que possibilitou uma revolução da informação. No entanto, a coleta em tempo real e o processamento de dados pessoais e de consumo dos usuários por grandes empresas de publicidade e software levanta dúvidas sobre a legitimidade e legalidade de como os dados pessoais dos indivíduos tem sido tratados. Por conta disso, é natural que os governos se voltem a aprovar novas legislações que definam e reforcem os direitos dos seus cidadãos. A implantação da *General Data Protection Regulation* (GDPR) foi um marco para as legislações de proteção de dados ao definir regras rígidas e inspirar outros governos a fazer o mesmo, como o governo brasileiro, que aprovou a Lei Geral de Proteção de Dados (LGPD). Em especial, definiu que os usuários tenham o direito de determinar para que propósitos os seus dados pessoais podem ser processados. Neste trabalho, investigamos a implicação desse direito no controle de acesso dos Sistemas de Gerenciamento de Banco de Dados (SGBDs), e propomos a extensão da Linguagem de Consulta Estruturada (SQL) para permitir o gerenciamento de propósitos intrinsecamente nesses sistemas.

Palavras-chave: SGBD. Propósito. Proteção de dados. SQL. Processamento de Consulta.

ABSTRACT

With the global population increasingly connected to the internet and the growing use of information systems in services and processes, there has never been so much data available to be processed, analyzed and researched, which enabled an information revolution. However, the real-time collection and processing of users' personal data and consumer custom by giant advertising and software companies raises doubts about the legitimacy and legality of how individuals' personal data has been handled. Because of this, it is natural that governments turn their attention to approving new legislation that defines and reinforces their citizens rights. The GDPR's implementation was a milestone for data security legislation, that defined more strict rules and inspired other governments to do the same, such as the Brazilian government, which approved the LGPD. GDPR defined, in particular, that users have the right to determine for what purposes their personal data may be processed. In this work, we investigate the implication of this right in DBMSs' access control, and we propose the extension of SQL to allow purpose management in DBMSs.

Keywords: DBMS. Purpose. Data Protection. SQL. Query Processing.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

GDPR	*General Data Protection Regulation*
LGPD	Lei Geral de Proteção de Dados
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Linguagem de Consulta Estruturada
CCPA	*California Consumer Privacy Act*
UE	União Européia

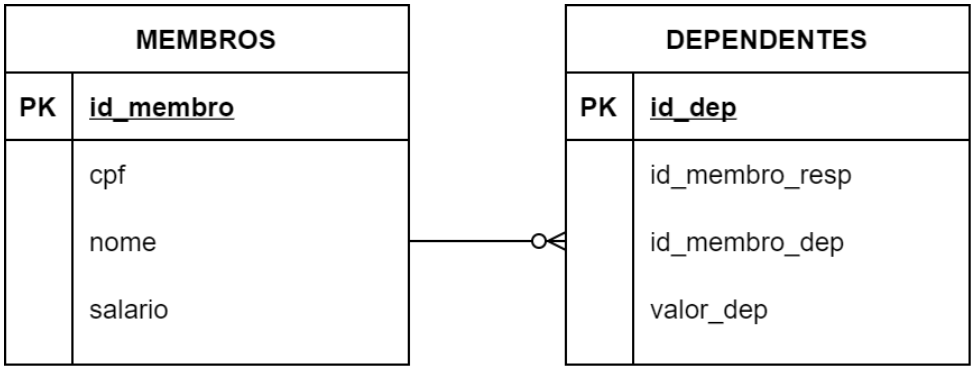
SUMÁRIO

1 INTRODUÇÃO

Embora o crescente volume de dados de usuário em posse das empresas de tecnologia possibilita o funcionamento de sistemas nunca vistos antes, como sistemas que executam algoritmos de aprendizagem de máquina em tempo real sobre dados de milhões, e possivelmente bilhões, de pessoas. Recentemente, podemos observar uma crescente desconfiança nessas grandes empresas de software, e muitos se perguntam: As pessoas têm realmente controle sobre seus próprios dados pessoais? (SOLOVE; CITRON, 2017).

Para entendermos melhor a situação, suponha que um administrador de banco de dados (DBA) de uma instituição sabe que certo setor de pesquisa deseja acessar dados dos membros da instituição presentes na relação Membros, cujo diagrama de Entidade-Relacionamento está apresentado na Figura 1, com o intuito de realizar uma pesquisa estatística.

Figura 1 – Diagrama Entidade-Relacionamento das relações a serem usadas no exemplo.



Fonte: Elaborada pelo autor.

Suponha ainda que a instituição tem um setor de recursos humanos que precisa ter acesso ao número de dependentes dos membros da instituição para cálculo de remuneração. A relação entre Membros e Dependentes também está apresentada na figura. Os dados armazenados pertencem a quem? Aos membros, isto é, aos sujeitos a quem se referem, ou a instituição, que coletou e armazena os dados? Os sujeitos tem algum direito ao controle desses dados? A instituição pode processar e distribuir esses dados como bem entender?

Tendo em vista esse cenário, governos do mundo inteiro mostraram preocupação e se prontificaram a realizar estudos com o objetivo de atualizar as legislações

responsáveis por normatizar o direito de acesso, coleta, armazenamento e processamento de dados dos seus cidadãos. Algumas das legislações chamam atenção são: a General Data Protection Regulation (2016), a *California Consumer Privacy Act* (CCPA) (CALIFORNIA, 2018) e a LGPD (BRASIL, 2018). A GDPR ganhou relevância por definir regras com tamanha rigidez, que influenciou outras legislações pelo mundo. No Brasil, a LGPD é um bom exemplo. Ela guarda muitas semelhanças à GDPR.

1.0.1 Regulamento Geral de Proteção de Dados - GDPR

A General Data Protection Regulation (GDPR) (General Data Protection Regulation, 2016) é, possivelmente, a mais rigorosa lei de privacidade e segurança de dados do mundo. Implementada pela União Européia (UE) no dia 25 de maio de 2018, ela impõe obrigações a organizações em todo lugar, bastando que essa colete ou deseje acessar dados referente a pessoas da UE (AG, 2021). Essa legislação define três indivíduos principais. O *data subject*, ou detentor dos dados, é a pessoa a quem os dados se referem. O *controller*, ou controlador, que é a empresa ou instituição que possui a infraestrutura que armazena e gerencia os dados. E o *processor*, ou processador, que é o individuo, empresa ou instituição que deseja realizar computação sobre os dados.

Segundo a GDPR, dados pessoais devem:

1. Ser processados legalmente, de forma justa e transparente com o detentor dos dados;
2. Coletados com propósitos específicos, explícitos e legítimos e não podem ser processados de maneira incompatível com esses propósitos;
3. Estar adequados e limitados ao que for necessário em relação aos propósitos para os quais são processados;
4. Acurados e atualizados;
5. Possibilitar a identificação do sujeito a que se referem não mais do que o necessário para o cumprimento dos propósitos para os quais são processados;
6. Processados de maneira a assegurar a segurança apropriada dos dados pessoais, incluindo proteção contra processamento desautorizado ou ilegal e contra perda acidental, destruição ou danos.

Além disso, salientamos que, segundo a GDPR, o controlador deve ser

responsável e, ser capaz de demonstrar conformidade com os princípios acima.

Dito isso, a legislação define que o processamento de dados pessoais só é considerado legal se respeitar pelo menos um das seguintes condições:

1. O sujeito detentor dos dados deu consentimento para que seus dados pessoais sejam processados para propósitos específicos;
2. O processamento é necessário para estabelecimento de um contrato em que o sujeito faz parte;
3. O processamento é necessário para cumprir uma obrigação legal do *data subject*;
4. O processamento é necessário para proteger interesses vitais do sujeito detentor dos dados ou de outra pessoa;
5. O processamento é necessário para a realização de uma tarefa necessária para cumprir o interesse público ou no exercício da autoridade oficial do controlador;
6. O processamento é necessário para alcançar interesses legítimos do controlador ou de um terceiro, exceto em situações em que esses interesses são sobrepostos pelos direitos fundamentais e liberdades ou interesses do *data subject* que necessitam da proteção de dados pessoais, em particular quando o detentor dos dados é menor de idade.

1.0.2 Lei Geral de Proteção de Dados Pessoais - LGPD

A Lei Geral de Proteção de Dados Pessoais (LGPD) (BRASIL, 2018), por sua vez, é uma norma brasileira de 2018 cujo número da lei é 13.709 que entrou em vigor no mês de agosto desse ano. Essa norma prescreve as medidas que as organizações que lidam com dados pessoais devem respeitar ao realizar o gerenciamento dos dados que estão sob sua posse. A LGPD surgiu diante da necessidade de garantir a proteção de dados dos usuários, seguindo o fluxo que estava ocorrendo em outros países, por exemplo o advento da GDPR que foi discutida acima. Na realidade, há uma semelhança entre os dispositivos da LGPD com a GDPR visto que a lei brasileira tomou como base os dispositivos que já existiam na época em que foi iniciada o seu processo de criação.

Semelhante à GDPR, a LGPD estabelece também, em seu art. 6º, que o tratamento de dados pessoais deve observar a boa-fé e dez princípios fundamentais específicos (CGU, 2020). O processamento de dados deve:

– Ser realizado com propósitos legítimos, específicos e explícitos, informados ao

titular e sem possibilidade de tratamento; posterior infingindo essas finalidades, ou seja, eles não podem ser adquiridos para um propósito, armazenados e então utilizados para uma finalidade que não tenha consentimento do detentor dos dados;

- Ser compatível com as finalidades informadas ao detentor dos dados;
- Ser limitado ao mínimo para o cumprimento de suas finalidades;
- Garantir aos detentores dos dados consulta facilitada e gratuita a respeito da forma e duração do tratamento e a integralidade dos seus dados;
- Garantir exatidão, clareza e relevância no cumprimento da finalidade consentida;
- Garantir informações claras, precisas e de fácil acesso sobre sua realização, respeitando os segredos comerciais e industriais;
- Proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas de destruição, perda, alteração, comunicação ou difusão, utilizando medidas técnicas e administrativas;
- Adotar medidas de prevenção de ocorrência de danos oriundos do tratamento;
- Ser impedido de ser realizado para fins discriminatórios ilícitos ou abusivos; e
- Adotar de medidas eficazes que comprovem a observância e cumprimento das normas de proteção de dados.

Observando a lista acima, é possível notar que, novamente, há a restrição de que os dados em posse de uma dada organização só poderão ser usados por ela mediante o consentimento, ou autorização, do seu respectivo dono para propósitos legítimos e específicos.

Com base nessas legislações, definimos que:

1.0.3 Propósito

Definição 1: Dados privados são instâncias de dados dos quais é possível obter informações sensíveis a respeito do indivíduo a que se referem.

Definição 2: Propósito é a finalidade para a qual determinada instância de dado privado será, ou pode ser, processada, como para publicidade ou análise estatística, por exemplo.

Definição 3: Consentimento é a concessão de permissão para processamento de uma determinada instância de dado para um determinado propósito.

Definição 4: Gerenciamento de propósito é um conjunto de mecanismos que modela lógica e fisicamente o consentimento, através da qual é possível identificar para quais propósitos cada instância de dado privado pode ser acessada.

Definição 5: Reforço de consentimento ou reforço de propósito é um conjunto de procedimentos que visam garantir que o consentimento é respeitado, isto é, que dados privados só poderão ser acessados perante apresentação de um propósito válido e previamente consentido para cada instância específica de dado.

1.0.4 Motivação

Hoje, empresas que armazenam e processam dados pessoais já podem receber, e estão recebendo, multas milionárias pelo não cumprimento dessas normas. Na Europa, as multas ultrapassam 300 milhões de euros. Visto isso, a adequação dos sistemas as legislações de proteção de dados é mandatório.

Voltando ao exemplo, podemos perceber que, com base nessas legislações, os setores de pesquisa e de recursos humanos da instituição são considerados processadores perante a lei e devem declarar explicitamente o propósito pelo qual desejam acessar e processar os dados pessoais dos membros. Já esses, são considerados detentores de dados e podem declarar que consentem com o uso de seus dados para esses propósitos se desejarem. Assim o setor de recursos humanos deve declarar que deseja acessar os dados com o propósito de calculo de remuneração, e o de pesquisa, realizar pesquisa estatística e aprendizado de máquina, por exemplo. Os dados poderão ser acessados, portanto, se o sujeito consentiu em permitir o acesso a eles especificamente para esses propósitos. A entidade que gerencia o banco de dados é o controlador e deve garantir que isso ocorra.

Como os Sistemas Gerenciadores de Banco de Dados (SGBDs) são, justamente, os responsáveis por gerenciar o armazenamento de dados no banco, eles estão entre os primeiros sistemas a serem afetados por essas legislações. Se os SGBDs forem capazes de fazer o controle dos propósitos do acesso aos dados e do consentimento dos detentores de dados, os sistemas e aplicações que os utilizam não precisarão se preocupar em implementar esse controle. A partir disso, observamos que é necessário o desenvolvimento de mecanismos, nos SGBDs, que permitam isso. A SQL, do inglês Structured Query Language, já permite especificar algum tipo de

controle de acesso a dados do banco, mas de maneira implícita. De forma que esta, apresenta poucas maneiras eficientes de lidar com o consentimento requisitado por essas legislações.

Observe a consulta abaixo sobre a tabela do exemplo anterior:

```
SELECT * FROM Membros;
```

Essa consulta normalmente retorna todas as tuplas da tabela Membros. No entanto, como a tabela membros possui dados pessoais, o SGBD deve processá-la de maneira diferente. Primeiro o usuário que realiza a consulta deve explicitar, de alguma forma, o propósito dessa consulta. E então, o SGBD deve retornar somente as tuplas cujos sujeitos a quem os dados se referem tenham consentido previamente com o propósito declarado.

Primeiro, é necessário atrelar a consultas ao banco de dados uma representação do propósito dessa. Segundo, é necessária uma maneira de definir o consentimento prévio dos detentores de dados ao acesso aos seus dados pessoais para propósitos específicos, isto é, é preciso relacionar dados no banco a uma representação dos propósitos permitidos para acessá-los. Por fim é preciso que os propósitos das consultas sejam comparados aos propósitos relacionados a dados no banco, de modo que somente os dados permitidos sejam retornados como resultado da consulta.

Utilizando a SQL que está implementada em SGBDs comerciais, poderíamos tentar realizar esse controle utilizando funções de controle de permissões de acesso como *GRANT* e *REVOKE*, no entanto, na prática, essas permissões de acesso são diferentes do controle de propósito que desejamos implementar, e devem funcionar separadamente. As funções de permissão de acesso em SQL, em geral, servem para delimitar quais entidades do banco de dados um usuário ou grupo e usuários, *role*, pode acessar e quais operações da SQL ele pode realizar sobre esses dados. No entanto, uma vez que um usuário tem permissão para realizar operações a um conjunto de dados, ele pode fazer isso com diferentes intenções, ou seja, ter diferentes propósitos para o acesso e processamento desses dados. Além disso, se tentarmos utilizar esse mecanismo de controle de acesso para gerenciar propósitos, quando houver um número grande de propósitos possíveis para serem definidos e aplicados a uma grande quantidade de conjuntos diferentes de dados, teríamos muitas combinações de

roles e regras de restrição o que tornaria esse método demasiadamente complexo e impraticável de ser gerenciado. O controle de acesso, portanto, deve ser usado para garantir que as operações realizadas sobre o banco de dados são feitas pelas pessoas corretas e não é ideal para gerenciar restrições de consentimento. Por isso, a definição e controle de consentimento deve ser realizada em uma etapa diferente, e posterior, ao controle de permissão de acesso.

Neste trabalho, estamos interessados em estudar o controle do consentimento, isto é, como garantir que os dados de um usuário serão processados somente para os fins específicos com os quais o individuo consentiu ou deu permissão de forma explicita. O principal objetivo desse trabalho é estudar e avaliar uma estratégia de especificação de consentimento para construir um mecanismo que conceda aos usuários do SGBD o controle sobre seus dados pessoais por meio da extensão da gramática SQL, fornecendo mecanismos intuitivos de reforço do direito de consentimento de forma explicita.

Este trabalho tem como objetivos específicos:

1. Propor regras gramaticais que permitam estender a linguagem SQL para simplificar a interação do usuário do SGBD com o gerenciamento de propósitos.
2. Propor uma forma de vincular dados de tabelas, colunas e tuplas a propósitos para garantir consentimento.

As contribuições do trabalho são (1) uma extensão da gramática SQL para definição de consentimento com propósitos; (2) algoritmos para a definição e gerenciamento de propósitos pelos usuários; e (3) a implementação da extensão da gramática e da definição de propósitos em um SGBD relacional.

Como contribuição científica desse trabalho, um artigo com os resultados alcançados foi submetido para avaliação no Simpósio Brasileiro de Banco de Dados - SBBD que irá ocorrer no segundo semestre do ano de 2021. No momento atual, estamos aguardando o resultado dessa apreciação.

Este trabalho está organizado da seguinte forma: No Capítulo 2 explicamos como as consultas são processadas em um SGBD e discutimos as propostas de gerenciamento de propósitos encontradas na literatura. No Capítulo 3, apresentamos propostas para a garantia de consentimento em SGBDs. Na Seção 3.1, apresentamos a proposta de extensão dos SGBDs relacionais para processamento de consultas

com restrição de propósitos. Na Seção 3.2, propomos 9 comandos que possibilitam uma iteração com as definições de propósito por meio de linguagem declarativa. Em seguida, apresentamos a extensão da gramática da linguagem SQL para aceitação dos comandos de gerenciamento de propósito no início do Capítulo 4. Ainda nesse capítulo, descrevemos os algoritmos desenvolvidos para executar os novos comandos. Por fim, no Capítulo 5 fazemos as considerações finais e levantamos ideias de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas de Gerenciamento de Banco de Dados

Um SGBD é um *software* projetado para auxiliar a manutenção e utilização de grandes volumes de dados. A alternativa aos uso de SGBDs é o armazenamento em arquivos e a escrita de uma aplicação com código específico para gerenciá-los. No entanto, o uso de SGBDs tem numerosas vantagens importantes (RAMAKRISHNAN *et al.*, 2003, pg. 4). Uma delas é a possibilidade de manipular os dados através de uma linguagem declarativa, a linguagem SQL. Isso significa que o usuário só precisa descrever qual o resultado que ele deseja receber como saída, ficando por conta do SGBD decidir quais dados acessar, o modo como isso ocorrerá, a ordem e como esses dados serão processados. Neste trabalho, focamos nosso estudo nos SGBDs relacionais, isto é, sistemas que organizam e manipulam dados estruturados em tabelas, as quais cada linha ou tupla representa um objeto, e cada coluna ou atributo, uma informação a respeito desse objeto. As tabelas contendo dados são agrupadas em um esquema, que é utilizado para gerenciar as tabelas e funções do banco de dados. O esquema contém, além das tabelas de dados, tabelas de sistema e outras informações essenciais para o funcionamento do SGBD. Para acessar um SGBD relacional é necessário abrir uma conexão com a interface do banco e definir o esquema com o qual o usuário deseja interagir. Após cada consulta, é atualizado um *log* que permite auditar as operações feitas sobre o banco de dados e recuperar o banco de danos eventuais.

Mais especificamente, esse trabalho está focado no processamento de consultas que é o módulo responsável por realmente receber as consultas submetidas e realizar todo o processamento necessário para obter o resultado da consulta.

2.1.1 Processamento de consulta

O módulo de processamento de consulta é um dos mais complexos dos SGBDs. Em virtude disso, o processamento de consulta é dividido em quatro fases principais que podem ser resumidas da seguinte forma:

Na primeira fase, a consulta precisa ser compreendida pelo módulo de *parser*, isto é, ela precisa estar escrita de acordo com a gramática aceita pelo SGBD,

a SQL. Assim, nessa fase é realizado tanto uma análise léxica quanto uma análise sintática da consulta, resultando na geração de uma árvore de consulta. Esta é uma arvore sintática em que cada nó representa uma operação relacional, como junção, seleção, etc. Resumindo, nessa primeira fase, é verificado se a consulta está de acordo com a linguagem esperada pelo SGBD.

Após isso, na segunda fase, o SGBD começa a realizar uma busca pela forma mais otimizada de executar aquela árvore gerada no passo anterior. Inicialmente, a consulta é reescrita usando as diversas equivalências da álgebra relacional. Posteriormente, a consulta já reescrita é repassada ao módulo de *planner*. Nesse módulo, o SGBD gera um conjunto de possíveis planos de execução equivalentes da consulta. Neste tipo de plano já se sabe quais operações serão realizadas, assim como quais algoritmos serão utilizados para processar essas operações. Por exemplo, os nós folhas, que representam as operações de leitura das relações envolvidas na consulta, já são associados a um método de acesso específico, como busca de índice ou varredura sequencial. Em resumo, essa etapa realiza a primeira fase da otimização das consultas: reescreve e gera um conjunto de planos de execução equivalentes.

A terceira fase é realizada já no contexto do módulo *optimizer* dos SGBDs. Neste módulo, é realizado a avaliação daqueles planos de execução gerados no passo anterior e, para avaliá-los, calcula uma estimativa do custo de execução de cada um deles. Com essas informações de custo, o SGBD escolhe o plano de execução com o menor custo calculado e o submete ao módulo *executor*.

Por fim, a última fase é realizada no módulo *executor* que é responsável por receber o plano de execução otimizado na fase anterior e realizar a execução desse plano. Isto é, esse módulo decodificará o plano de execução recebido, o executará e, finalmente, retornará os dados requeridos pela consulta submetida ao SGBD.

Neste artigo, nos limitamos a discutir como garantir, no âmbito dos SGBDs, que em geral fazem o papel de *controller*, que os dados pessoais do *data subject* armazenados no banco de dados só poderão ser acessados com propósitos específicos definidos explicitamente previamente consentidos pelo detentor dos dados.

2.2 Gerenciamento de Propósitos em SGBD

A verificação de propósitos correspondentes a tuplas é uma instância do problema clássico de pertinência de conjuntos, já explorado em outros trabalhos, como (RIZVI *et al.*, 2004) que descreve um modelo de autorização onde consultas são reescritas. Neste trabalho, o controle de acesso é feito através do uso de visões de autorização que é basicamente uma visão tradicional de banco de dados que possui alguns parâmetros de controle em sua definição, tais como identificador de usuário, localização de usuário, entre outros. As consultas são reescritas para que essas visões sejam inseridas no momento da execução das consultas. Assim, um controle de acesso fino pode ser realizado.

Já com o conceito de propósito em mente, os autores de (BYUN; LI, 2008) propuseram tratar propósitos como entidades hierárquicas, estabelecendo operações para a transição entre estes. Isto é, os autores definiram propósitos de modo a utilizar essa noção para guiar a execução das consultas. Para isso, os autores propuseram técnicas para realizar tanto a criação quanto o controle dos propósitos em diferentes níveis de granularidade. Por fim, é importante dizer que a ideia de usar propósitos em hierarquia permitiu que as técnicas propostas pelos autores fosse escalável.

Recentemente, o tema voltou ao foco por conta do surgimento de diversas legislações orientadas à preservação da privacidade de indivíduos e a proibição do uso irrestrito de seus dados. Por exemplo, (PAPPACHAN *et al.*, 2020) define um *middleware* para processamento de consultas orientadas a propósito para possibilitar o controle fino de acesso escalável em ambientes que necessite de um grande número de políticas para definir para quais propósitos os dados podem ser acessados e processados. Os autores apresentaram duas técnicas para realizar esse controle fino, além de demonstrar formalmente o funcionamento de ambas. A primeira técnica é responsável por realizar, inicialmente, um filtro de quais as políticas importantes para uma dada consulta. Posteriormente, a segunda técnica é utilizada para realizar um filtro de tuplas, ou seja, quais as tuplas que satisfazem as políticas descobertas no passo anterior. Assim, ao usar essas duas técnicas, é possível reduzir tanto o número de consultas a ser verificadas quanto o número de tuplas que precisam satisfazer cada política. Por fim, essa última técnica que realiza a filtragem das tuplas foi baseada em uma técnica anterior bastante aplicada na área de banco de dados.

(WANG *et al.*, 2019) descrevem um paradigma para coletar, processar e gerenciar dados sensíveis e que visa garantir, além do consentimento, transparência e auditoria, portabilidade dos dados e a não identificação do indivíduo, facilitando a adequação a legislações como a GDPR. Esse sistema consiste em quatro conseitos principais: as capsulas (*data capsules*), programas de análise, grafo de capsulas e gerenciador de *data capsule*. As cápsulas combina dados privados, uma política que governa o uso dos dados e metadados que descrevem as propriedades da capsula e dos dados presentes nela. Os programas de análise são procedimentos que processam as capsulas e retornam uma nova capsula e o gerenciador de *data capsule* rastreia as capsulas através do grafo e capsulas. Com esse paradigma, os dados só podem ser desencapsulados, isto é, desvinculados das políticas, através de um procedimento específico. Além disso, nesse trabalho foi feito uma codificação para a GDPR utilizando a ferramenta Legalease que é utilizada para analisar estaticamente os programas de análise, e também realizaram experimento para avaliar o desempenho da proposta. Nesse trabalho, as politicas incluem os propósitos.

(SHASTRI *et al.*, 2020) especifica um conjunto de propriedades que os SGBDs precisam atender para satisfazer a GDPR. Nesse trabalho, é abordado um fenômeno chamado de explosão de metadados. Ao definir metadados de propósito para os dados privados no banco, esses metadados podem causar um aumento significativo de mais de 3.5 vezes do armazenamento ocupado pelo banco, além disso, essa grande quantidade de dados causa atrasos durante a execução, pois eles precisam ser carregados para a memória e analisados para validar o acesso e, também, precisam ser escritos no *log* para validação futura. Esse trabalho também propõe um novo *benchmark* para quantificar a adequação a GDPR.

Já (MACHADO; AMORA, 2020) elabora sobre os impactos causados pela legislação, além de fazer uma revisão da literatura e apontar oportunidades de pesquisa. Esse trabalho discute a explosão de metadados e defende que o acesso com base em propósitos é necessário para garantir direito de acesso do sujeito aos seus dados, direito de ser informado sobre acessos aos seus dados e o direito ao consentimento. Também observa que a carga de trabalho dos SGBDs vai mudar consideravelmente uma vez que consultas de somente leitura passam a ser *write-intensive*.

(AGRAWAL *et al.*, 2005) e (PUN, 2010) atacam diretamente o problema da

extensibilidade do SQL para a definição de propósitos de formas distintas. O trabalho (AGRAWAL *et al.*, 2005) define um modelo de restrições semelhante a permissões, associando o acesso a um conjunto de permissões e reescrevendo a consulta de forma a manter o estado do banco em relação às restrições. Para isso, os autores propuseram uma nova sintaxe que permite a criação de restrições de propósitos com diferentes granularidades: linha, coluna e célula. Além disso, é possível definir para que tipos de comandos os propósitos devem ser respeitos: *select*, *delete*, *insert*, *update* ou todos. Por fim, propuseram também a semântica quando há várias restrições ao mesmo tempo juntamente com um algoritmo para isso.

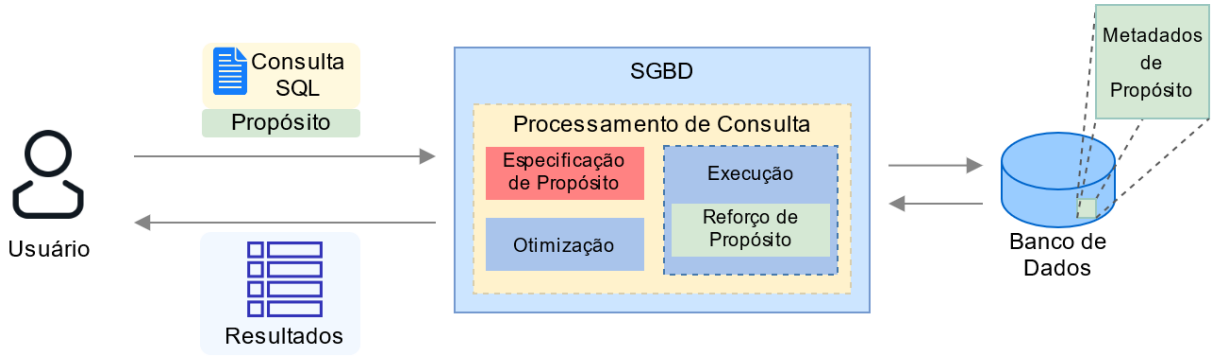
O último trabalho relacionado é o (PUN, 2010) que propôs alterar as cláusulas padrão SQL para adicionar funções de generalização e validação aos dados, enquanto outras cláusulas passam a exigir de forma explícita o propósito de acesso. Assim como o trabalho relacionado anterior, esse também propôs uma nova extensão para lidar com a introdução do conceito de propósitos nas consultas SQL. Assim, também é possível definir a granularidade da aplicação dos propósitos no momento da execução das consultas. No entanto, a abordagem desses autores é baseado nos comandos de *GRANT*/*REVOKE*, que com a extensão proposta agora possuem um campo que especifica para qual propósito aquele comando está sendo executado. Também apresentaram a semântica desses comandos estendidos. Por fim, os autores analisaram a complexidade introduzida por essa extensão proposta.

Este trabalho se diferencia por propor modificações intrínsecas para garantir consentimento por meio de gerenciamento de metadados de propósito nos SGBDs, sem o uso de programas externos, buscando uma implementação direta no processamento de consultas do sistema, que parte desde o módulo de *parser*, com a extensão da gramática, até o módulo de execução. Nossa pesquisa visa possibilitar os SGBD a gerenciar os propósitos de forma eficiente e modular.

3 METODOLOGIA

3.1 Sistema para Processamento de Consultas Orientado a Propósito

Figura 2 – Fluxo de consultas com consentimentos/propósitos dos usuários.



Fonte: Elaborada pelo autor.

Quando um usuário submete uma consulta SQL a um SGBD relacional, o processamento de consultas tradicional consiste em realizar as etapas de reconhecimento, otimização e execução para gerar a resposta correta (ELMASRI; NAVATHE, 2000). Neste tipo de arquitetura, a saida da consulta é uma tabela formada por todas as tuplas que satisfazem às condições impostas pela consulta, como, por exemplo, predicados de seleção e condições de junção. Ao analisarmos o direito ao consentimento, definidos pela LGPD e pela GDPR, identificamos uma nova restrição de acesso aos dados, o propósito da consulta. O propósito da consulta é uma representação explicita do motivo pelo qual o usuário deseja acessar e processar os dados da consulta. Resultados de consultas devem ser consistentes com o consentimento definido pelos detentores de dados pessoais presentes no banco de dados. Para possibilitar que SGBDs lidem com o conceito de consentimento, desenhamos uma arquitetura de processamento de consultas orientado a propósito mostrada na Figura 2.

No fluxo apresentado na Figura 2, o usuário encaminha além da consulta SQL, os propósitos da consulta. A sintaxe da consulta SQL, na interface do SGBD, não é alterada. Propomos a adição de metadados de propósito a consultas, automaticamente formados pelo perfil do usuário e informação a respeito da aplicação com a conexão ativa. Ademais, propomos a adição de dois novos módulos ao processamento de consulta: Especificação de Propósito e Reforço de Propósito. O reforço de propósito é um submódulo do módulo de Execução responsável por fazer a comparação em

tempo de execução e consulta dos propósitos da consulta com os propósitos atrelados aos dados no banco, garantindo que os dados retornados pela consulta são somente aqueles que possuem consentimento para os propósitos apresentados na consulta ou são dados não privados. Já o de especificação de propósito é encarregado de realizar todo o gerenciamento de propósitos, isto é, da criação, manutenção e manipulação dos metadados que relacionam propósitos a dados no banco.

Em nossa pesquisa, estamos desenvolvendo esse sistema de gerenciamento de propósito. Neste trabalho, nos focamos na apresentação de uma extensão a ser incorporada na gramática SQL para dar suporte ao conceito de propósito e uma proposta de implementação para o gerenciamento de metadados de propósito. O detalhamento completo da arquitetura, com reforço de consentimento em tempo de execução, e os resultados parciais de eficácia e desempenho do protótipo estão fora do escopo deste trabalho.

3.2 Comandos para Gerenciamento de Propósitos

Os propósitos associados a um dado no banco pode ser representado por uma estrutura de dados de conjunto. Essa estrutura modela a teoria axiomática de conjuntos e deve permitir operações entre conjuntos, como união ($\cup$) e diferença ($\setminus$) de conjuntos. Conjuntos podem ser implementados como tipo abstrado de dado, em linguagem C, de diversas maneiras eficientes. Por exemplo, podem ser implementados utilizando tabelas de dispersão, lista encadeada ou mesmo filtros probabilísticos. Propomos a manutenção de uma estrutura de dados interna do SGBD, atrelado ao esquema, que armazena os possíveis propósitos na forma de conjunto. Como mostrado por Kraska *et al.* (2019), esta maneira possibilita a redução do espaço do armazenamento ocupado pelos metadados de propósito. Propomos a manutenção de uma segunda estrutura nos metadados do esquema que guarde para cada relação com dados privados um conjunto de propósitos atrelado a ela. Além disso, propomos que cada relação que possua dados privados armazene os seguintes metadados de propósito: uma estrutura que relaciona cada tupla a um conjunto de propósitos e uma estrutura de dados que relacione cada coluna da tabela a um conjunto de propósitos. Os elementos desses conjuntos devem pertencer ao conjunto de propósitos possíveis do esquema.

Dado isto, para permitir que o usuário do SGBD gerencie esses metada-

dos com facilidade através de uma linguagem declarativa, propomos os seguintes comandos:

CREATE PURPOSE p { ON SCHEMA s } (1)

UPDATE PURPOSE p TO new_name (2)

DROP PURPOSE p { ON SCHEMA s } (3)

SET PURPOSE p TO TABLE t (4)

SET PURPOSE p TO ROWS ON TABLE t { WHERE $P(x)$ } (5)

SET PURPOSE p TO COLUMN c ON TABLE t (6)

DELETE PURPOSE p FROM TABLE t (7)

DELETE PURPOSE p FROM ROWS ON table t { WHERE $P(x)$ } (8)

DELETE PURPOSE p FROM COLUMN c ON table t (9)

Os comandos de (1) a (3) permitem, respectivamente, a inclusão, substituição e remoção de elementos no conjunto de possíveis propósitos do esquema; Os comandos (4) e (7), permitem a inclusão e remoção de um propósito ao conjunto de propósitos de uma relação; Os comandos (5) e (8), ao conjunto de propósitos relacionado a uma tupla, e os comandos (6) e (9), ao conjunto de propósitos de uma coluna. A seguir, descrevemos os comandos com mais detalhes.

Com o primeiro comando (1), o usuário do SGBD insere um nome p para o propósito. Se um nome de esquema s for inserido, o SGBD cria um propósito com nome p e adiciona-o ao conjunto de propósitos do esquema s, caso contrário, o propósito é adicionado ao conjunto de propósitos do esquema da conexão atual do SGBD. As chaves representam uma entrada opcional. Com o comando (2), o SGBD verifica se o propósito com nome p pertence ao conjunto de propósitos do esquema e, caso pertença, ele será renomeado para *new_name*, no entanto, todas as referencias a ele em relações, colunas e tuplas continuam a se referir a ele sem necessidade de um procedimento em cascata. Por fim, com o comando (3), um propósito p e todas as suas referências são excluídos em cascata do esquema correspondente.

O comando (4) é utilizado para fazer a ligação entre um propósito com nome p e uma relação t. Ao receber esse comando, o SGBD adiciona p ao conjunto de propósito da relação t. Já o comando (5) faz a associação do propósito de nome p a todas as tuplas x na relação t em que a proposição lógica $P(x)$ for verdadeira. Caso não seja inserida uma cáusula $P(x)$, o propósito será aplicado a todas as tuplas por

vacuidade. O SGBD, então, insere p ao conjunto de propósitos atrelado a x. O comando (6), por sua vez, adiciona p ao conjunto referente ao atributo c armazenado na estrutura de dados presente nos metadados da tabela t. Os comandos (7), (8) e (9) são inversos aos comandos (4), (5) e (6) respectivamente. Removendo o propósito com nome p das respectivas estruturas de dados.

Escolhemos esses símbolos não terminais para se assemelhar com a gramática de comandos presentes na SQL. Observe que enquanto o comando (1), (2) e (3) se assemelham aos comandos *CREATE TABLE*, *DROP TABLE* e *UPDATE TABLE*, que manipulam um objeto no banco de dados. Já os comandos que começam com *SET* criam ligações entre esses objetos, os propósitos, a outros objetos no banco, isto é, a tuplas, colunas e tabelas. E os comandos que começam com *REMOVE* desfazem essas ligações.

Para demonstrar como esses comandos podem ser utilizados, vamos voltar para o exemplo apresentado no começo desse trabalho. Considere as tabelas da figura 1. Para garantir o direito de consentimento dos membros da instituição, tendo acesso aos comandos propostos, o DBA insere os seguintes comandos no SGBD:

```
CREATE PURPOSE "Pesquisas Estatísticas e Aprendizado de Máquina"
CREATE PURPOSE "Cálculo de Remuneração"
```

Ao fazer isso, serão alocados identificadores para os propósitos “Pesquisas Estatísticas e Aprendizado de Máquina” e “Cálculo de Remuneração”, por exemplo, “P1” e “P2”, respectivamente, como pode ser observado na parte superior da Figura 3 que apresenta como os metadados serão armazenados. Nesse momento, o SGBD tem o conhecimento da existência dos dois referidos propósitos e, com isso, pode ser associado aos dados. Para isso, os seguintes comandos devem ser executados:

```
SET PURPOSE "Cálculo de Remuneração" TO COLUMN salario ON TABLE
Membros
SET PURPOSE "Cálculo de Remuneração" TO COLUMN valor_dep ON TABLE
Dependentes
```

Ao executar os dois comandos acima, o SGBD estará realizando a as-

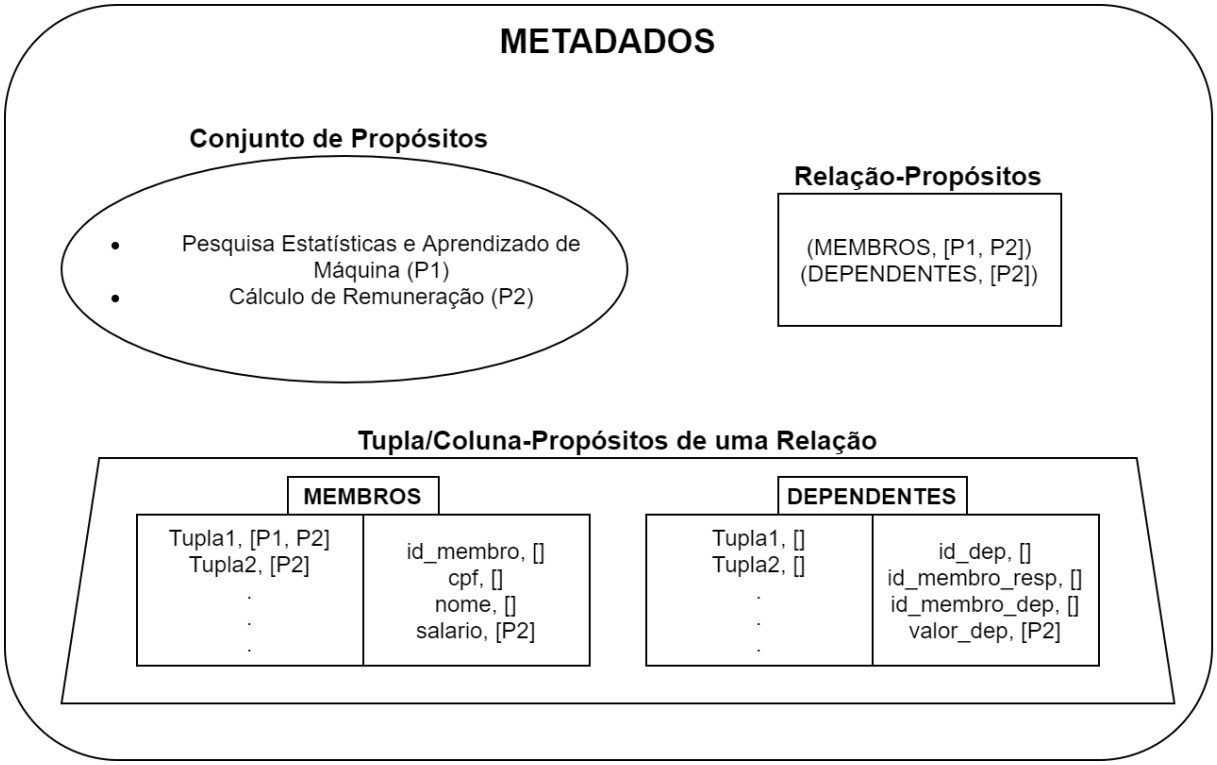
sociação entre o propósito de "Cálculo de Remuneração"e as relações Membros e Dependentes, o que significa dizer que existem tuplas ou colunas ou a própria relação inteira que devem ser processados somente para aqueles dados propósitos. No lado direito superior da Figura 3, os metadados responsáveis por garantir que essa informação seja salva são mostrados. Ou seja, pares de (Relação, Propósitos) são armazenados.

Além disso, como os comandos acima não envolvem a relação em si, mas uma das suas colunas, então outros metadados são armazenados. No entanto, nesse caso, esses metadados serão específicos por relação e, portanto, estarão localizados juntamente com a relação a que se refere. Veja a parte inferior da Figura 3 que mostra o efeito dos comandos acima na criação dos metadados das duas relações envolvidas.

Por fim, é possível que cada membro pode então optar por compartilhar seus dados para a pesquisa através do comando:

```
SET PURPOSE "Pesquisas Estatísticas e Aprendizado de Máquina" TO ROWS
ON TABLE Membros AS m WHERE m.cpf = <cpf>
```

Figura 3 – Metadados gerados no exemplo descrito.



Fonte: Elaborada pelo autor.

Novamente, esse comando irá gerar novas entradas de metadados na parte dos metadados específicos das relações. Por exemplo, ao inserir esse comando, suponha que a tupla em que o valor na coluna “cpf” for igual a <cpf> é a tupla $Tupla1$ de Membros. Assim, será incluída uma nova entrada nos metadados de $P2$. Observe que na Figura 3 a $Tupla1$ de Membros possui dois propósitos atrelados: $P1$ e $P2$. O primeiro por causa do atual comando que descrevemos e o segundo por causa do propósito atrelado à coluna $salrio$ de Membros que foi adicionado ao executar os comandos descritos acima. No entanto, veja que a $Tupla2$ só possui o propósito $P2$ atrelado.

Com isso, alcançamos o cenário de que somente tuplas com o propósito “Pesquisas Estatísticas e Aprendizado de Máquina” devem estar disponíveis para consultas do setor de pesquisas que visam realizar operações estatísticas sobre os dados de membros da instituição. O processador de consulta, por meio do processo de geração do plano de execução, vai automaticamente garantir essa restrição na geração do resultado. O detalhamento desse processo está fora do escopo deste trabalho, mas tem como base outras abordagens descritas em Kraska *et al.* (2019) e Pappachan *et al.* (2020), onde a omissão de tuplas é uma estratégia comum, mas incluem informações adicionais no resultado como a quantidade de tuplas omitidas por não satisfazer as propósitos de acesso.

Com base nessas propostas, estendemos a implementação da linguagem SQL de um SGBD comercial para aceitar os comandos propostos. Escolhemos realizar a implementação sobre o PostgreSQL, um SGBD relacional de código aberto (*Open Source*) implementado em linguagem C. Fizemos essa escolha pois a linguagem C é amplamente difundida e esse SGBD possui uma documentação completa disponível publicamente, além disso, sua licença de código aberto possui poucas restrições e nos permite, no âmbito desse trabalho, modificar o seu código fonte sem violá-la.

4 RESULTADOS

Neste capítulo, apresentamos detalhes da implementação das propostas do trabalho no PostgreSQL. Na Seção 4.1, apresentamos como estender a SQL para produzir os comandos propostos; Por fim, na Seção 4.2, apresentamos algoritmos para a execução dos comandos.

4.1 Gramática de Gerenciamento de Propósitos

Para que a SQL possa produzir os comandos apresentados, precisamos que o módulo *parser* do SGBD aceite os comandos como parte da linguagem SQL, para isso são necessários os seguintes passos.

4.1.1 Análise Léxica

O analisador léxico do parser deve reconhecer os novos símbolos terminais e produzir os comandos. Portanto, adicionamos o símbolo terminal PURPOSE, que é produzido ao encontrar a cadeia de caracteres "purpose", seja em letras maiúsculas ou minúsculas. Os demais símbolos terminais *CREATE, UPDATE, DROP, SET, DELETE, ON, TO, TABLE, SCHEMA, ROWS, FROM e COLUMN*, necessários para produzir os comandos de gerenciamento de propósito, já estão presentes na lista de símbolos da SQL.

Após isso, adicionamos à gramática da SQL as produções apresentadas na Figura 4. O símbolo não-terminal *stmt* é o símbolo inicial da gramática SQL, responsável por produzir todos as possíveis consultas, como criação de tabelas e seleções. Representamos isso com reticências (...). Nossa gramática acrescenta cinco novas produções ao símbolo inicial *stmt*, são elas: *CreatePurposeStmt*, *UpdatePurposeStmt*, *DropPurposeStmt*, *SetPurposeStmt* e *DeletePurposeStmt*. Essas produções serão responsáveis por produzir os comandos propostos iniciadas por CREATE, UPDATE, DROP e SET, respectivamente. Feito isso, como a SQL já possui símbolos *table_ref* e *columnref* que reconhecem nomes de tabelas e de colunas respectivamente, basta acrescentar um símbolo que reconhece nomes de propósitos, que denominamos *purpose_ref*. O símbolo *SCONST* reconhece cadeia de caracteres. Por fim, o simbolo *where_clause_p* se baseia na cláusula *where_clause* presente na gramática da seleção, mas diferente

Figura 4 – Gramática da extensão proposta para a SQL usando o Formalismo de Backus-Naur (BNF).

```
⟨stmt⟩                    ::= ...
                            | ⟨CreatePurposeStmt⟩
                            | ⟨UpdatePurposeStmt⟩
                            | ⟨DropPurposeStmt⟩
                            | ⟨SetPurposeStmt⟩
                            | ⟨DeletePurposeStmt⟩

⟨CreatePurposeStmt⟩       ::= CREATE PURPOSE ⟨purpose_ref⟩
                            | CREATE PURPOSE ⟨purpose_ref⟩ ON SCHEMA ⟨name⟩

⟨UpdatePurposeStmt⟩       ::= UPDATE PURPOSE ⟨purpose_ref⟩ TO ⟨Sconst⟩

⟨DropPurposeStmt⟩         ::= DROP PURPOSE ⟨purpose_ref⟩
                            | CREATE PURPOSE ⟨purpose_ref⟩ ON SCHEMA ⟨name⟩

⟨SetPurposeStmt⟩          ::= SET PURPOSE ⟨purpose_ref⟩ TO TABLE ⟨table_ref⟩
                            | SET PURPOSE ⟨purpose_ref⟩ TO ROWS ON TABLE ⟨table_ref⟩
                            | SET PURPOSE ⟨purpose_ref⟩ TO ROWS ON TABLE ⟨table_ref⟩ ⟨where_clause_p⟩
                            | SET PURPOSE ⟨purpose_ref⟩ TO COLUMN ⟨columnref⟩ ON TABLE ⟨table_ref⟩

⟨DeletePurposeStmt⟩       ::= DELETE PURPOSE ⟨purpose_ref⟩ FROM TABLE t ⟨statement⟩
                            | DELETE PURPOSE ⟨purpose_ref⟩ FROM ROWS ON TABLE ⟨table_ref⟩
                            | DELETE PURPOSE ⟨purpose_ref⟩ FROM ROWS ON TABLE ⟨table_ref⟩
                              ⟨where_clause_p⟩
                            | DELETE PURPOSE ⟨purpose_ref⟩ FROM COLUMN ⟨columnref⟩ ON TABLE
                              ⟨table_ref⟩

⟨purpose_ref⟩             ::= ⟨SCONST⟩
```

Fonte: Elaborada pelo autor.

dessa que possui produções complexas como seleções aninhadas, *where_clause_p* só aceita as cláusulas lógicas (como por exemplo $table\_alias.column\_name = value$) representadas por *Expr*.

4.1.2 Geração da Árvore Sintática Abstrata

Em seguida, programamos a geração das arvores sintáticas abstratas para as produções. Para isso, criamos novos nós de *parse* (*parse nodes*) que codificam os comandos. Esses nós servem de entrada para o Planner. Como esses comandos não precisam ser otimizados, adicionamos o rótulo *CMD_UTILITY*. Quando o *planner* recebe uma instrução com esse rótulo, ele empacota-a e repassa para o *executor*. A partir disso, basta implementar a execução desses comandos para que o SGBD esteja pronto para gerenciar metadados de propósitos.

4.2 Execução dos Comandos

Nessa Seção, discutiremos como esses comandos podem ser processados para refletir na manipulação eficiente dos metadados.

Para executar o comando *CreatePurposeStmt*, propomos o Algoritmo 1. O primeiro passo consiste em identificar o esquema (*schema*) a partir do nome do esquema guardado na estrutura *CreatePurposeStmt*. Se o valor do nome do esquema for nulo, o que significa que o usuário não inseriu o nome do esquema, recupere o esquema da conexão atual do cliente. O passo seguinte é verificar se existe, no esquema, o conjunto P de propósitos possíveis. Se não existir, é preciso criar um conjunto P vazio. O próximo passo é criar um par $new\_purpose$ que representa o novo propósito. O par $new\_purpose$ identifica o nome $purpose\_ref$ inserido pelo usuário a um identificador numérico único que será utilizado pelo SGBD para procedimentos internos. Por fim, basta inserir o par $new\_purpose$ no conjunto P.

Algoritmo 1: Algoritmo CreatePurpose

Entrada: Estrutura CreatePurposeStmt $node$ contendo referência do propósito $purpose\_ref$ e o nome do esquema $schema\_name$
Saída: inteiro positivo id do propósito

```
1  início
2      schema ← getSchema(node.schema_name);
3      se schema é nulo então
4          schema ← getCurrentSchema()
5      fim
6      P ← getPurposes(schema) se P é nulo então
7          P ← {}
8      fim
9      id ← generateId() new_purpose ← (id, node.purpose_ref);
10     P ← P ∪ {new_purpose};
11     retorna id;
12 fim
```

Os comandos *UpdatePurposeStmt* e *DropPurposeStmt* são bem semelhantes ao comando *CreatePurposeStmt*, e podem ser vistos nos Algoritmos 2 e 3, respectivamente. Para executar UpdatePurposeStmt, assim como naquele comando, o primeiro passo é identificar o esquema. Em seguida, verificar se o conjunto P de propósitos do esquema existe, caso contrário, retorna uma mensagem de erro e a operação é

abortada. Se existir, o algoritmo verifica se o propósito de nome $purpose\_ref$ pertence a P. Se o elemento não for encontrado, a operação não pode ser concluída e uma mensagem de erro notifica o usuário do ocorrido. Caso o elemento seja encontrado ele é removido do conjunto e substituído pelo novo par formado por new_name e pelo identificador de $purpose\_ref$.

Algoritmo 2: Algoritmo UpdatePurpose

Entrada: Estrutura UpdatePurposeStmt $node$ contendo referência do propósito $purpose\_ref$ e o novo nome new_name
Saída: inteiro positivo id do propósito

```
1  início
2      schema ← getCurrentSchema();
3      P ← getPurposes(schema);
4      se P é nulo então
5          Imprima mensagem de erro;
6          retorna nulo;
7      fim
8      id ← getId(P, purpose_ref);
9      se id é nulo então  Imprime mensagem de erro; retorna nulo ;
10     purpose ← (id, node.purpose_ref) ;
11     new_purpose ← (id, node.new_name);
12     P ← P \ {purpose};
13     P ← P ∪ {new_purpose};
14     retorna id;
15 fim
```

Já na construção *DropPurposeStmt*, assim como nos comandos acima, o algoritmo também recupera o esquema e verifica a existência de P. Se o propósito $purpose\_ref$ pertencer ao conjunto P, remova $purpose\_ref$ desse conjunto. Em seguida, é preciso percorrer os metadados de propósito de tabelas, colunas e tuplas, e remover em cascata o propósito $purpose\_ref$ de todos os elementos. Outra solução seria inserir o propósito $purpose\_ref$ a um conjunto "lixo", que identifica que este propósito foi excluído do SGBD, e realizar a remoção de $purpose\_ref$ nas demais estruturas a medida que o SGBD processa consultas a essas tabelas, colunas e tuplas. No entanto, essa abordagem precisa ser investigada mais a fundo.

A implementação da produção *SetPurposeStmt* pode ser observada no Algoritmo 4. Após recuperar o esquema e o conjunto P, o algoritmo acessa o campo

Algoritmo 3: Algoritmo DropPurpose

Entrada: Estrutura DropPurposeStmt *node* contendo referência do propósito *purpose_ref* e o nome do esquema *schema_name*

Saída: Verdadero em caso de sucesso, e falso caso contrário

```
1  início
2      schema ← getSchema(node.schema_name);
3      se schema é nulo então
4          schema ← getCurrentSchema()
5      fim
6      P ← getPurposes(schema);
7      se P é nulo então
8          Imprima mensagem de erro;
9          retorna falso;
10     fim
11     pid ← getId(node.purpose_ref);
12     purpose ← (pid, node.purpose_ref);
13     H ← getMap(schema);
14     se P contém purpose então
15         H ← getPurposeMap(schema);
16         se H é não nulo então
17             para cada par (t, P_t) em H faça
18                 V_t ← getTablePurposeArray(t);
19                 h_t ← getTablePurposeMap();
20                 para cada atributo c em t faça V_t[c] ← V_t[c] \ {pid} ;
21                 para cada par (r, h_t[r]) em h_t faça
22                     h_t[r] ← h_t[r] \ {pid} ;
23                     se h_t[r] está vazio então h_t ← h_t \ {(r, h_t[r])};
24                 fim
25                 P_t ← P_t \ {pid};
26                 se P_t é nulo então H ← H \ (t, pid) ;
27             fim
28         fim
29         P ← P \ {purpose};
30         retorna verdadeiro;
31     senão
32         retorna falso
33     fim
34 fim
```

Algoritmo 4: Algoritmo SetPurpose

Entrada: Estrutura SetPurposeStmt $node$ contendo referência do propósito $purpose\_ref$, o rótulo do comando cmd, a expressão lógica $expr$, e o indetificador da coluna $column\_ref$ e o identificador de tabela $table\_ref$

Saída: Verdadeiro ou Falso

```
1  início
2      schema ← getCurrentSchema()
3      P ← getPurposes(schema)
4      se P é nulo então
5          Imprima mensagem de erro;
6          retorna falso;
7      fim
8      id ← getId(P, purpose_ref);
9      se id é nulo então
10         Imprima mensagem de erro;
11         retorna falso;
12     fim
13     selecione cmd faça
14         caso 4 faça
15             H ← getPurposeMap(schema);
16             se H é nulo então
17                 H ← {};
18             fim
19             table_purpose ← H[table_ref];
20             se table_purpose é não nulo então
21                 table_purpose.value ← table_purpose.value ∪ {id} ;
22                 H ← H \ table_purpose ;
23                 H ← H ∪ table_purpose ;
24             senão
25                 H ← H ∪ (table_ref, id) ;
26             fim
27         fim
```

```
28
    /* continuação do algoritmo 4                                          */
29
        caso 5
            h_t ← getTableMap(schema,table_ref);
            se h_t é nulo então
                h_t ← {};
            fim
            se expr é não nulo então
                t = select * from table_ref where expr;
            senão
                t = select * from table_ref;
            fim
            para cada r em t faça
                tuple_id ← getId(table_ref, r);
                r_purpose ← h_t[tuple_id)] se r_purposenulo então
                    h_t ← h_t ∪ (tuple_id, purpose_ref);
                senão
                    new_tuple_purpose ← (tuple_id, r_purpose ∪ purpose_ref);
                    h_t ← {h_t \ (tuple_id, r_purpose)};
                    h_t ← {h_t ∪ new_tuple_purpose};
                fim
            fim
        fim
        caso 6
            V_t ← getTableMap(schema,table_ref);
            se V_t é nulo então
                Instancia um vetor V_t[getColumnNumber(table_ref)];
                para i de 0 a getColumnNumber() faça
                    V_t[i] = {};
                fim
                V_t[getColumnId(column_ref)] ←
                  V_t[getColumnId(column_ref)] ∪ purpose_ref;
            fim
        fim
        outro
            Imprima mensagem de erro;
            retorna falso;
        fim
    fim
    retorna verdadeiro
fim
```

cmd da estrutura *SetPurposeStmt* e identifica de qual comando ela se trata. Caso se trate do comando (4), o passos seguintes consistem em: recuperar o identificador $purpose\_id$ único do propósito $purpose\_ref$, se $purpose\_ref$ não pertence a P a operação falha; Verificar se existe o mapa H, caso não exista, criar um mapa H que relaciona tabelas a um conjunto de propósitos utilizando o identificador único da tabela como chave e o identificador do propósito como valor. Se a chave $table\_ref$ já pertence a H, adicionamos $purpose\_id$ ao conjunto de propósitos de $table\_ref$. Caso contrário, inserimos a chave $table\_ref$ com o valor $purpose\_id$ ao conjunto de propósitos da tabela $table\_ref$ no mapa H.

No caso do comando (5), é criado, caso não exista, um mapa h_t (lê-se "h"de "t"), para a relação $table\_ref$, que é armazenada nos metadados dessa relação. Caso $expr$ seja diferente de nulo, o algoritmo lê a tabela e faz uma varredura. Para cada tupla x que satisfaça a cláusula $expr$, caso x não esteja em h_t, o identificador $purpose\_id$ do propósito $purpose\_ref$ é recuperado e o par <x, $purpose\_id$> é inserido no mapa h_t, caso contrário, seja <x, p'> o par presente em h_t, com p' o conjunto de propósitos atrelados a x, adicionamos $purpose\_id$ ao conjunto p'. É importante deixar claro que há somente um mapa H no esquema do banco de dados, no entanto, haverá um mapa h_t para cada relação com dados privados.

No caso do comando (6), queremos associar um propósito $purpose\_ref$ para um atributo $column\_ref$. Para isso, o algoritmo cria, caso não exista, um vetor V_t cujos valores nas posições do vetor recebem o valor inicial nulo (∅). Cada posição representa um atributo da tabela $table\_ref$, e guarda-o junto aos metadados de $table\_ref$. Em seguida adicione o identificador $purpose\_id$ ao conjunto de propósitos p' na posição referente a coluna $column\_ref$ no vetor V_t.

Por fim, o comandos (7), (8) e (9) são inversos aos comandos (4), (5) e (6) respectivamente. Ou seja, em vez de inserir, removemos o identificador do propósito $purpose\_ref$ do conjunto de propósitos da relação, tupla ou coluna. No entanto, caso a remoção do identificador $purpose\_id$ resulte no conjunto vazio, removemos a relação $table\_ref$ de H em (7) e removemos a tupla x de h_t em (8). No entanto, no comando (9), o valor de V_t na posição referente a coluna $column\_ref$, nesse caso, recebe uma representação do conjunto vazio.

Implementamos a extensão da gramática no SGBD PostgreSQL e imple-

mentamos parcialmente os algoritmos que gerenciam os propósitos. Até a publicação dessa monografia, conseguimos definir e gerenciar propósitos para tabelas e colunas no banco de dados. Os comandos são reconhecidos pelo *parser* do SGBD. Inicialmente, utilizamos tabelas de dispersão para representar o conjuntos de propósitos possíveis do esquema e um vetor de lista de inteiros para cada tabela com dados privados para o conjuntos que relaciona colunas a propósitos. O gerenciamento de propósitos para tuplas está em desenvolvimento. Para os conjuntos de propósitos para tuplas, atualmente, estudamos uma implementação mais eficiente utilizando filtros probabilísticos. Além disso, estamos desenvolvendo as outras fazes do sistema, a saber, o reforço de propósito e a definição de propósitos para consultas. Em um futuro próximo, buscamos realizar experimentos para avaliar a eficiência prática desse sistema.

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho apresenta uma extensão da gramática da SQL que viabiliza o suporte ao restrições de acesso a dados pessoais por meio do consentimento a propósitos, trazido pelas novas normas de proteção e privacidade de dados. Descrevemos uma proposta de extensão da linguagem SQL por meio de 9 comandos adicionais para as produções da gramática da linguagem. Apresentamos a implementação dessa gramática estendida e exemplificamos como essas modificações podem ser utilizadas. A partir do exemplo, podemos observar que a manipulação dos propósitos se dá de forma simples e intuitiva para um usuário habituado com a linguagem SQL.

Como trabalhos futuros é possivel estudar:

- A utilidade e viabilidade de um aninhamento de comandos de gerenciamento de propósitos com outros comandos da linguagem SQL;
- Otimizações para os algoritmos de gerenciamento de propósitos, em especial, estudar uma implementação do vinculo de tuplas com propósitos realizando verificando as cláusulas em uma estrutura de índice;
- Como armazenar e gerenciar metadados que definam propósitos para células de relações, tornando a granularidade das regras de propósito ainda mais fina;
- Por fim, como garantir a conformidade com o direito de consentimento durante a execução de consultas SQL, isto é, como avaliar por exemplo, em tempo de execução, quais serão as tuplas, cujos propósitos não são violados pela consulta, que serão retornadas.

REFERÊNCIAS

AG, P. T. **GDPR.eu**. 2021. GDPR.EU is a website operated by Proton Technologies AG, which is co-funded by Project REP-791727-1 of the Horizon 2020 Framework Programme of the European Union. Disponível em: https://gdpr.eu/tag/gdpr/. Acesso em: 08 jun. 2021.

AGRAWAL, R.; BIRD, P.; GRANDISON, T.; KIERNAN, J.; LOGAN, S.; RJAIBI, W. Extending relational database systems to automatically enforce privacy policies. In: **ICDE**. [*S. l.*]: IEEE Computer Society, 2005. p. 1013–1022.

BRASIL. **Lei Nº 13.709 - Lei Geral de Proteção de Dados Pessoais (LGPD)**. 2018. http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acessado em: 01-07-2021.

BYUN, J.; LI, N. Purpose based access control for privacy protection in relational database systems. **VLDB J.**, v. 17, n. 4, p. 603–619, 2008.

CALIFORNIA. **California Consumer Privacy Act (CCPA)**. 2018. https://www.caprivacy.org/. Acessado em: 01-07-2021.

CGU. **GUIA DE BOAS PRÁTICAS - LEI GERAL DE PROTEÇÃO DE DADOS (LGPD)**. 2020. https://repositorio.cgu.gov.br/bitstream/1/44262/12/Guia-lgpd.pdf. Acessado em: 01-07-2021.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems, 3rd Edition**. [*S. l.*]: Addison-Wesley-Longman, 2000. ISBN 978-0-8053-1755-8.

General Data Protection Regulation. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. **Official Journal of the European Union**, v. 59, p. 1–88, 2016.

KRASKA, T.; STONEBRAKER, M.; BRODIE, M. L.; SERVAN-SCHREIBER, S.; WEITZNER, D. J. SchengenDB: A data protection database proposal. In: **Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2019 Workshops, Poly and DMAH, Los Angeles, CA, USA, August 30, 2019**. [*S. l.*]: Springer, 2019. (Lecture Notes in Computer Science, v. 11721), p. 24–38.

MACHADO, J. C.; AMORA, P. R. P. How can db systems be ready for privacy regulations. In: **SBBD**. [*S. l.*]: SBC, 2020.

PAPPACHAN, P.; YUS, R.; MEHROTRA, S.; FREYTAG, J. Sieve: A middleware approach to scalable access control for database management systems. **Proc. VLDB Endow.**, v. 13, n. 11, p. 2424–2437, 2020.

PUN, S. **Prisql: a privacy preserving sql language**. PRISM, 2010. Disponível em: https://prism.ucalgary.ca/handle/1880/104364.

RAMAKRISHNAN, R.; GEHRKE, J.; GEHRKE, J. **Database management systems**. [*S. l.*]: McGraw-Hill New York, 2003. v. 3.

RIZVI, S.; MENDELZON, A. O.; SUDARSHAN, S.; ROY, P. Extending query rewriting techniques for fine-grained access control. In: **SIGMOD Conference**. [*S. l.*]: ACM, 2004. p. 551–562.

SHASTRI, S.; BANAKAR, V.; WASSERMAN, M.; KUMAR, A.; CHIDAMBARAM, V. Understanding and benchmarking the impact of GDPR on database systems. **Proc. VLDB Endow.**, v. 13, n. 7, p. 1064–1077, 2020.

SOLOVE, D. J.; CITRON, D. K. Risk and anxiety: A theory of data-breach harms. **Tex. L. Rev.**, HeinOnline, v. 96, p. 737, 2017.

WANG, L.; NEAR, J. P.; SOMANI, N.; GAO, P.; LOW, A.; DAO, D.; SONG, D. Data capsule: A new paradigm for automatic compliance with data privacy regulations. **CoRR**, abs/1909.00077, 2019.