Extending the Integrity Constraint Support in a Graph Database

Fábio Reina, Alexis Huf, Daniel Presser and Frank Siqueira

***Abstract*—The large volume of data that has been produced every day has encouraged the development of new data storage technologies, including graph databases, which have experienced a growing adoption rate in recent years. On the other hand, the graph database management systems that are commercially available nowadays still lack means to ensure the consistency of stored data. In this scenario, this work proposes the extension of a graph database management system in order to add support for four new integrity constraints, allowing the definition of: conditions on node attributes, required edges, type of in/out nodes of an edge and edge cardinality. Support for these constraints was added to a modified version of OrientDB and experimental results show improvements in the performance of applications with integrity constraints.**

***Index Terms*—Graph databases, integrity constraints, data consistency, data integrity, OrientDB.**

I. Introdução

Devido à constante evolução da tecnologia de informação, foi observado nos últimos anos um grande crescimento do volume de dados produzidos e armazenados por sistemas computacionais. Esse fenômeno, que ficou conhecido como *Big Data* [1], é hoje abordado em uma quantidade significativa de trabalhos de pesquisa, o que vem fomentando o desenvolvimento de novos sistemas de armazenamento e análise de dados. Devido a essa tendência, indústria e academia têm realizado esforços em busca de soluções para diversas limitações advindas da necessidade de manipular grandes massas de dados.

Juntamente com o crescimento no volume de dados, a busca por melhorias no desempenho de consultas e de gerenciamento mais eficiente de bancos de dados (BDs) vem resultando em uma intensa evolução tecnológica [2]. Contudo, os sistemas de gerenciamento de banco de dados (SGBDs) tradicionais nem sempre conseguem atender todos os requisitos de aplicações que manipulam grandes volumes de dados [3]. Como resultado, uma nova classe de SGBDs, denominados *Not only SQL* (NoSQL), surgiu com o objetivo de suprir essas necessidades. A principal característica dos bancos NoSQL é o relaxamento da consistência do dado em busca de ganhos de desempenho. Esse aspecto permite que o dado fique inconsistente por um certo momento, o que pode ser proibitivo para algumas aplicações que dependem desses BDs.

Quando os bancos de dados de grafo (BDGs) oferecem algum suporte para especificação de restrições de integridade (RIs), em geral as RIs suportadas são bastante limitadas [4]. As restrições mais comuns permitem aos administradores impor que atributos tenham valores únicos, definir valores máximos e mínimos, e associar tipos a atributos [5]. Devido à falta de suporte para RIs mais complexas, a validação de dados normalmente se torna responsabilidade da aplicação que utiliza o BDG, exigindo maior esforço em seu desenvolvimento. Nos casos em que essa verificação é omitida, o grafo pode alcançar um estado inconsistente.

O presente trabalho visa minimizar essa limitação existente em BDGs propondo a adição de suporte para a definição e verificação de RIs. Dessa forma, pretende-se transferir a responsabilidade pela consistência do dado da aplicação para o BDG, auxiliando a obtenção da integridade do dado e diminuindo o esforço no desenvolvimento de aplicações. Para alcançar este objetivo, foram especificados e implementados quatro novos tipos de RIs em um BDG: restrição condicional para a propriedade do nó, restrição de arestas obrigatórias, restrição de tipo dos nós de entrada e saída de uma aresta, e restrição de cardinalidade de relações. Por fim, foi avaliado o impacto dessas novas restrições sobre o tempo de execução das operações do BDG.

O restante deste trabalho está organizado como segue. Na Seção II são apresentados os conceitos mais relevantes usados na pesquisa. Na Seção III é descrita a proposta de adição de suporte a RIs em um BDG. Na sequência, na Seção IV é apresentado o estudo realizado para avaliação do impacto da adição de suporte a novas RIs ao BDG. Em seguida, na Seção V, trabalhos similares encontrados na literatura são comparados com a solução proposta neste artigo. Por fim, na Seção VI são apresentadas as conclusões alcançadas com o desenvolvimento deste trabalho e são identificados alguns problemas em aberto que devem ser objeto de trabalhos futuros.

II. Fundamentação Teórica

Um BDG é um BD que usa a estrutura explícita de um grafo para armazenar, consultar e manipular dados. Nessa estrutura, os vértices (também chamados de nós) representam os registros do BD, enquanto arestas representam as relações entre os dados [4]. Em geral, toda aresta tem um rótulo que identifica o tipo de relacionamento que ela representa. Assim como vértices, arestas também podem ter propriedades, o que as torna tão importantes quanto estes, devido às informações que representam. Em razão da composição, a estrutura é dita ser um grafo de propriedades [6]. Este trabalho adota a definição de grafo de propriedade apresentada na Def. 1.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brasil - Código de Financiamento 001, e da Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC).

Esta definição foi adaptada de [7] a fim de remover múltiplos rótulos e valores múltiplos de um mesmo atributo, o que melhor se alinha às implementações atuais de SGBDs de grafo, incluindo Neo4J, OrientDB, InfinityGraph, Titan e ArangoDB.

Def. 1: Um grafo de propriedade é uma tupla $G = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, tal que:

1) N é um conjunto de nós;
2) E é um conjunto de arestas;
3) $\rho : E \to (N \times N)$ é uma função que associa uma aresta em E a um par (o, t) de nós de origem e destino em N;
4) $\lambda_N : N \to L_N$ é uma função que associa um nó a um rótulo de nó;
5) $\lambda_E : E \to L_E$ é uma função que associa uma aresta a um rótulo de aresta;
6) $\sigma : (N \cup E) \times P \to V$ é uma função que dado um nó N ou aresta E, junto a uma propriedade P, associa o par a um valor V.

Grafos são modelos mais naturais para representar e visualizar alguns tipos de dados, como aqueles com grande número de relacionamentos. A representação de dados altamente conectados em um modelo relacional é possível, mas resulta em diversas relações muitos-para-muitos. Uma consulta nessas condições requer um grande número de junções, degradando o desempenho do BD [8]. Em contrapartida, a estrutura de grafo proporciona um melhor gerenciamento desse tipo de dado, resultando na execução mais rápida de consultas [9]. O exemplo mais recorrente de aplicação que utiliza esse tipo de banco são as redes sociais; no entanto, BDGs também são muito utilizados em sistemas financeiros, para monitoramento de transações e detecção de fraudes [8], [10]; no setor de varejo, ajudando na decisão de compra e recomendação de produtos [11]; além de diversas outras aplicações.

Diferentemente dos BDs relacionais, que possuem um *schema* bem definido, os BDGs não possuem esse tipo de estrutura rígida. Essa característica resulta na execução mais rápida das operações de manipulação de dados. Por outro lado, fica mais difícil impor regras de integridade, já que não há um *schema* a seguir. Como resultado, os BDGs em geral não fornecem meios para garantir a consistência dos dados ou suportam apenas RIs muito simples.

BDGs são categorizados sob um grupo maior, denominado BDs NoSQL. Dentre suas características gerais, pode-se destacar as interfaces simples e flexíveis para consulta e manipulação de dados. BDs NoSQL seguem as propriedades BASE, do inglês *Basically Available, Soft state, and Eventual consistency*, que buscam priorizar a execução de operações de leitura e escrita em detrimento da consistência dos dados. Dessa forma, os dados podem ficar inconsistentes por um certo tempo, se tornando íntegros em um momento futuro [12].

Consistência, em BDs com suporte às propriedades BASE, refere-se a transações que executam leituras em dados atualizados e salvos [12]. A corretude dos dados não é uma preocupação de consistência, mas uma preocupação de integridade dos dados. Portanto a integridade dos dados pode ser definida como a garantia de que o dado estará correto durante todo seu ciclo de vida [13]. Dessa forma, integridade dos dados é um aspecto crítico do projeto, implementação e uso de qualquer sistema, o que significa que sua ausência pode ser proibitiva. Ela pode ser alcançada pelo uso de um conjunto de regras que especificam as modificações permitidas nos dados [4]. Consequentemente, essas regras definem atualizações válidas nos dados e em seus relacionamentos e devem ser aplicadas a todos os dados recém-inseridos para evitar inconsistências. Se a integridade não puder ser garantida pelo SGBD, sua garantia deverá ser implementada no nível da aplicação caso necessário.

Nesse cenário, RIs passaram a ser levadas em consideração em SGBDs dos mais diversos modelos de dados, para que possam garantir a consistência e a integridade dos dados perante as RIs definidas [4]. O uso de RIs garante que a informação armazenada no banco está correta em respeito a estas regras e deixa o dado mais próximo de alcançar a integridade. No entanto, a maioria das implementações de BDGs disponíveis no mercado não suporta RIs, ou permite apenas a definição de regras simples, incapazes de fornecer o nível de integridade exigido pela maioria das aplicações que armazenam e manipulam dados na forma de grafo.

III. Extensão do Suporte a RIs em Grafos

São inúmeras as aplicações que podem se beneficiar do uso de BDGs. No entanto, a falta de suporte à definição de RIs acaba sendo um impedimento para aplicações que exigem maior acurácia dos dados. Conforme já mencionado, muitos dos SGBDs disponíveis no mercado capazes de representar dados na forma de grafos não permitem a declaração de RIs, e aqueles que as suportam, implementam apenas restrições simples. Portanto, este trabalho consiste em estender a implementação de um SGBD de grafo a fim de permitir a definição e a verificação de RIs complexas, úteis para aplicações que representam dados na forma de grafo e necessitam de um maior nível de integridade.

O SGBD escolhido para o desenvolvimento desse trabalho foi o OrientDB [14]. Dentre as características que motivaram sua escolha está o fato de este SGBD já oferecer suporte nativo para definição de algumas RIs simples. Além disso, ele está entre os SGBDs mais populares com suporte para grafos, de acordo com o ranking DBEngines [15]. Apesar deste trabalho utilizar o OrientDB, a sua implementação também é possível em outros BDGs.

No OrientDB, cada classe (também denominada tipo de nó) está associada a um conjunto de metadados. Esses metadados foram estendidos para armazenar definições de RIs, que descrevem a restrição usando a extensão da linguagem de definição de dados (DDL) do OrientDB. Depois que uma RI é definida, toda alteração no conjunto de instâncias dessa classe – isto é, a criação, atualização ou deleção de um nó – será validada com a RI antes da efetivação da transação. Por exemplo, uma restrição de intervalo, em que os valores de máximo e mínimo de uma propriedade são restringidos, irá verificar se o valor atribuído está dentro do intervalo permitido sempre que os valores de suas propriedades forem alterados. Todas as RIs suportadas adotam essa estratégia de validação antes de concluir uma transação.

As RIs suportadas nativamente pelo OrientDB estão restritas a comparar valores entre os dados que estão sendo inseridos ou atualizados com valores de restrição definidos nos metadados do banco, para o tipo de nó correspondente [16], [17]. Uma abordagem para implementar RIs mais complexas, não limitadas aos atributos do nó, é calcular dinamicamente os atributos do nó e compilar definições dessas RIs complexas em RIs mais simples, limitadas à verificação dos valores dos atributos do nó. No entanto, essa estratégia incorre em uma grande sobrecarga na forma de atributos dinâmicos e em um número potencialmente grande de metadados necessários para implementar as RIs. Para evitar essas desvantagens, a extensão proposta neste trabalho introduz um novo componente na arquitetura do OrientDB, o gerenciador de restrições. Este componente implementa a validação para novos tipos de RI diretamente, sem depender das restrições já suportadas pelo SGBD. O uso de um gerenciador de restrições dedicado permite RIs que também envolvem arestas, além de nós e propriedades. Sua existência também facilita a introdução de novos tipos de RI, fornecendo um único ponto de interceptação para sua validação. A extensão proposta neste trabalho introduz suporte para quatro novos tipos de RI: (i) Condicional; (ii) Aresta obrigatória; (iii) Entrada/Saída e (iv) Cardinalidade.

O gerenciador de restrições é inicializado junto com o servidor do OrientDB e usa uma *SB-Tree* [18] para armazenar e gerenciar objetos do tipo *Constraint*. Quando o usuário cria uma RI, o analisador extrai o tipo de restrição, seu destino (ou seja, classe de nó, propriedade ou classe de aresta) e argumentos específicos de restrição. O resultado da análise alimenta uma fábrica de restrições e o objeto de restrição resultante é serializado na *SB-Tree* do gerenciador de restrições.

Depois que a restrição é declarada e armazenada na *SB-Tree*, toda operação de gravação no BDG aciona a validação dos objetos *Constraint* que têm como alvo a propriedade e classe de nó ou de aresta mencionada na operação de criação da RI. Após localizar todos os objetos *Constraint* relevantes, o gerenciador de restrições coleta todos os dados importantes da própria operação de criação e chama o procedimento de validação de cada objeto. Se a validação de qualquer objeto *Constraint* falhar, uma exceção é lançada, levando a transação a ser abortada. Contudo o OrientDB não faz a revalidação dos dados pré-existentes e sim de novas entradas, portanto dados antigos ainda podem apresentar discordância quanto à RI.

A sintaxe de cada nova RI é apresentada no restante desta seção, junto com uma definição formal e com alguns exemplos que demonstram seu uso em cenários hipotéticos. Todas as definições usam o grafo de propriedade apresentado na Def. 1. Algumas RIs permitem operadores relacionais, que são denotados por $\alpha \in \mathcal{O}$, onde $\mathcal{O} = \{<, \leq, =, \neq, \geq, >\}$. Tipos de nó e aresta são denotados pela notação $\mathcal{T}_\beta$, onde $\beta \in L$ é um rótulo que identifica o tipo sem ambiguidade. Tipos são definidos formalmente pelos conjuntos de suas instâncias: $\mathcal{T}_\beta = \{x \mid \lambda(x) = \beta\}$. Assim, rótulos de nós e arestas são conjuntos disjuntos, $\beta \in L_N \iff \mathcal{T}_\beta \subseteq N$ e $\beta \in L_E \iff \mathcal{T}_\beta \subseteq E$.

Restrição Condicional (*Conditional*). O objetivo dessa restrição é comparar propriedades de um nó e validar valores atribuídos a elas de acordo com uma condição definida anteriormente. Portanto, essa RI é aplicada à classe do nó. Esta RI pode ser usada, por exemplo, para classificar nós que descrevem uma pessoa de acordo com a idade. Suponha que cada nó *Person* contenha as propriedades *age* e *category*. Se o valor de *age* for menor que 18, então *category* deve ser *minor*, caso contrário, deve ser *adult*. Sua definição formal é dada na Def. 2 e a gramática para a declaração dessa restrição é mostrada na Fig. 1(a), enquanto a Fig. 1(b) mostra a transcrição da restrição descrita no exemplo anterior. Sempre que uma transação tenta inserir um novo nó *Person* ou tenta atualizar as propriedades *age* ou *category* de um nó existente, os novos valores de *age* e *category* são confrontados com os valores especificados na restrição e, dependendo do resultado da comparação, o gerenciador de restrições decide se permitirá a execução da transação ou não.

Def. 2: Dado um grafo de propriedade $G = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, uma Restrição Condicional é uma tupla Cond = $(\mathcal{T}_o, p_1 \in P, \alpha_1 \in \mathcal{O}, v_1 \in V, p_2 \in P, \alpha_2 \in \mathcal{O}, v_2 \in V, p_3 \in P, \alpha_3 \in \mathcal{O}, v_3 \in V)$, tal que:

$$\forall o \in \mathcal{T}_o \; . \; \begin{cases} \sigma(o, p_2) \; \alpha_2 \; v_2, & \text{se } v_1 \in \sigma(o, p_1) \\ \sigma(o, p_3) \; \alpha_3 \; v_3, & \text{caso contrário.} \end{cases}$$

```
<CREATE> <CONSTRAINT> name
<ON> class <(> attribute <)>
<CONDITIONAL> <(>
  <IF> property (>|<|>=|<=|=|!=) expression
  <THEN> property (>|<|>=|<=|=|!=) expression
  [ <ELSE> property (>|<|>=|<=|=|!=) expression ]
<)>
```

(a) Sintaxe da Restrição Condicional

```
CREATE CONSTRAINT personCategory
ON Person (category)
CONDITIONAL (IF age < 18 THEN category = 'minor'
                         ELSE category = 'adult')
```

(b) Exemplo de Restrição Condicional

Fig. 1: Sintaxe (a) e exemplo (b) de Restrição Condicional.

Restrição Aresta Obrigatória (*Required Edge*). Esta RI define que uma classe de nó tenha um tipo de aresta de saída obrigatório, que apontará para uma instância de uma classe de nó de destino. Portanto, se houver um nó cuja classe apareça como classe de origem na restrição, deverá haver pelo menos uma aresta saindo desse nó e chegando a um nó com a classe de destino especificada. Os metadados da restrição estão associados à classe do nó de origem. Um caso de uso dessa restrição é um aplicativo de comércio eletrônico, no qual uma compra deve conter pelo menos um produto. Isso é representado pela classe *Order*, cujas instâncias devem ter pelo menos uma aresta *contains* conectando-as a um nó do tipo *Product*. A definição formal é apresentada na Def. 3 e sua sintaxe geral é mostrada na Fig. 2(a), enquanto a Fig. 2(b) mostra a definição da restrição correspondente ao exemplo descrito anteriormente.

Def. 3: Dado um grafo de propriedade $G = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, uma Restrição Aresta Obrigatória é uma tupla Req = $(\mathcal{T}_o, \mathcal{T}_e, \mathcal{T}_t)$ tal que: $\forall o \in \mathcal{T}_o \; : \; (\exists e, t \; : \; e \in \mathcal{T}_e \; \wedge \; t \in \mathcal{T}_t \wedge \rho(e) = (o, t))$.

```
<CREATE> <CONSTRAINT> name
<ON> origin_class <REQUIRED_EDGE>
    [edge_type] <TO> target_class
```

(a) Sintaxe da Restrição Aresta Obrigatória

```
CREATE CONSTRAINT orderHasProducts
ON Order REQUIRED_EDGE contains TO Product
```

(b) Exemplo de Restrição Aresta Obrigatória

Fig. 2: Sintaxe (a) e exemplo (b) da RI de Aresta Obrigatória.

Restrição Entrada/Saída (*In/Out*). Esse tipo de restrição restringe as classes de nós que participam como origem ou destino de uma classe de aresta. Ao mesmo tempo, é útil para impor a direção do relacionamento representado. Por exemplo, uma pessoa escreve um documento e não o contrário. Diferentemente de outras restrições nesta proposta, essa restrição é armazenada nos metadados associados à classe da aresta em vez da classe de nó. A Def. 4 apresenta a definição formal dessa RI e a sintaxe geral é mostrada na Fig. 3(a): a restrição é aplicada sobre uma classe de aresta e pelo menos uma das classes das extremidades dessa aresta deve ser especificada. Um possível caso de uso é restringir as arestas da classe *drives* para conectar apenas nós *Person* aos nós *Car*. Uma operação de inserção de "*John drives (a) horse*" seria rejeitada, bem como uma inserção de "*Mustang drives John*". A Fig. 3(b) mostra esses casos de uso modelados como restrições na DDL proposta.

Def. 4: Dado um grafo de propriedade $G = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, uma Restrição Entrada/Saída é uma tupla IO = $(\mathcal{T}_o, \mathcal{T}_e, \mathcal{T}_t)$ tal que: $\forall e \in \mathcal{T}_e \ : \ o \in \mathcal{T}_o \ \wedge \ t \in \mathcal{T}_t$, onde $\rho(e) = (o, t)$.

```
<CREATE> <CONSTRAINT> name
<ON> edge_type <IN_OUT_EDGE> (
       <FROM> origin_class <TO> target_class
     | <FROM> origin_class
     | <TO> target_class
)
```

(a) Sintaxe da Restrição Entrada/Saída

```
CREATE CONSTRAINT personDrives
ON drives IN_OUT_EDGE (
       FROM Person TO Car
)
```

(b) Exemplo da Restrição Entrada/Saída

Fig. 3: Sintaxe (a) e exemplo (b) da RI de Entrada/Saída.

Restrição de Cardinalidade (*Cardinality*). O objetivo é restringir o número de arestas de uma determinada classe que conecta um nó de origem aos nós de destino. A especificação de cardinalidade consiste em dois números inteiros, com *N* servindo como uma representação para "não especificado", ou seja, qualquer número é permitido. As restrições de cardinalidade também permitem a especificação de classes de nós. No entanto, estão associadas à classe do nó de origem e, portanto, não permitem uma classe de nó de origem não especificada. Sua definição formal é apresentada na Def. 5 e a Fig. 4(a) mostra a sintaxe geral desse tipo de restrição. A restrição está sempre associada à classe do nó de origem e à classe de aresta. Após a especificação da cardinalidade, pode-se opcionalmente especificar a classe do nó de destino.

Um exemplo de caso de uso é quando uma instituição de ensino deseja limitar o número de alunos em seus cursos e também o número de cursos nos quais um aluno pode se matricular. Dados os nós *Student* e *Course*, é possível restringir a classe de aresta *enrolled* para conectar nós *Student* a nós *Course* com cardinalidade *25..5*. Em outras palavras, um aluno pode se inscrever em até 5 cursos, enquanto cada curso pode ter no máximo 25 alunos matriculados. A Fig. 4(b) mostra uma especificação desse caso de uso.

Def. 5: Dado um grafo de propriedade $G = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, uma Restrição de Cardinalidade é uma tupla Card = $(\mathcal{T}_o, \mathcal{T}_e, \mathcal{T}_t, \alpha \in \mathcal{O}, v \in V)$ tal que: $\forall o \in \mathcal{T}_o \ : \ ||\{(e,t) \mid (e,t) \in (\mathcal{T}_e \times \mathcal{T}_t) \wedge \rho(e) = (o,t)\}|| \ \alpha \ v$.

```
<CREATE> <CONSTRAINT> name
<ON> origin_class <CARDINALITY>
   [edge_type] <(> <INT | N> <..> <INT | N> <)>
   [<TO> target_class ]
```

(a) Sintaxe da Restrição de Cardinalidade

```
CREATE CONSTRAINT studentEnrollment
ON Student CARDINALITY
  enrolled (25 .. 5) TO Course
```

(b) Exemplo de Restrição Cardinalidade

Fig. 4: Sintaxe (a) e exemplo (b) da RI de Cardinalidade.

IV. Avaliação

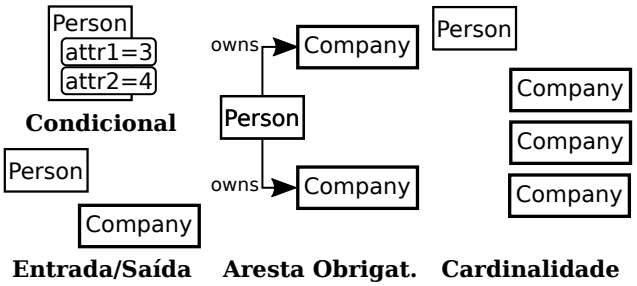
Os métodos de validação de RI propostos foram implementados no OrientDB 3.1.0. Para avaliar o impacto das alterações, o fator considerado foi o tempo de execução das operações. Portanto, essa avaliação se concentra no impacto das RIs sobre o desempenho das operações.

A estratégia adotada neste estudo de avaliação assume que RIs não podem ser ignoradas. Quando um desenvolvedor opta por utilizar um SGBD incapaz de impor RIs, a validação da restrição deve ser garantida por controles introduzidos na própria lógica da aplicação. Deste modo, os experimentos realizados foram construídos de forma a permitir a comparação de três variantes do OrientDB. A primeira consiste no servidor OrientDB *Original*, sem nenhuma modificação. A variante *Modificado* consiste em uma versão do OrientDB à qual foi adicionada o componente de validação de RIs proposto neste trabalho. Por fim, a variante *Aplicação* corresponde à execução da validação de RIs na aplicação cliente que envia comandos para o servidor OrientDB original.

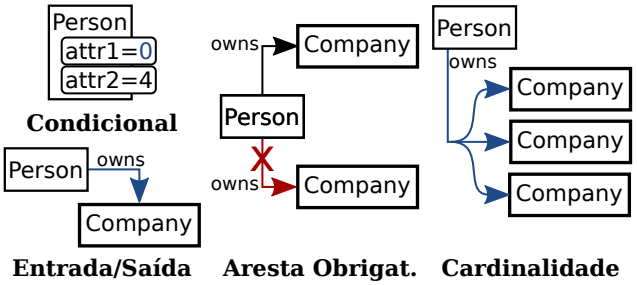
O código-fonte para reprodução dos experimentos está disponível em um repositório público [19]. Os experimentos foram executados de modo a evitar interferências causadas pela máquina virtual Java (JVM). Para isso, um novo par de JVMs é criado para cada medição: uma JVM executa o servidor OrientDB, enquanto a outra executa o JAR da aplicação de teste `constraints-tests-client`, que envia a transação e mede seu tempo de execução. Ambas as JVMs foram criadas no mesmo *host*, com sistema operacional Ubuntu 18.04.3 LTS 64 bits (kernel 5.0), executado em uma máquina com processador Intel i7-4510U dual-core a 2.0 GHz

```
1  CREATE CONSTRAINT cond ON Person
2  (attr1) CONDITIONAL (IF attr2 < 3 THEN attr1 < 2
3                                    ELSE attr1 > 4);
4
5  CREATE CONSTRAINT req ON Person
6  REQUIRED_EDGE owns TO Company;
7
8  CREATE CONSTRAINT inout ON owns
9  IN_OUT_EDGE FROM Person TO Company;
10
11 CREATE CONSTRAINT card ON Person
12 CARDINALITY owns N .. 3 TO Company;
```

Fig. 5: Restrições criadas durante os experimentos.



(a) Modelo para povoamento inicial do BD



(b) Modelo após a execução das transações

Fig. 6: Modelo do BD para avaliação das RIs.

com 16 GB de RAM. Uma única medição consiste em duas fases. Na primeira fase, o cliente configura o BD no OrientDB usando comandos de administração. Essa configuração contém as informações básicas do esquema, incluindo RIs e a população do BD, quando aplicável. Na segunda fase, 1000 transações análogas são executadas 3 vezes. O número de transações visa evitar a medição não confiável de operações rápidas e as 3 execuções visam evitar a interferência de tarefas em segundo plano não determinísticas, como GC (*Garbage Collector*), cache de disco e compilação JIT (*Just In Time*). Entre cada uma dessas 3 execuções, as caches de disco são liberadas (usando a chamada de sistema `sync()`) e o GC é solicitado a executar. Somente a terceira execução tem seu tempo registrado.

Foram projetados 4 cenários experimentais visando quantificar o *overhead* da validação de cada tipo de RI. A Fig. 5 apresenta as RIs criadas para cada caso. Cada cenário para validação de uma RI consiste de dois passos, ambos esquematizados na Fig. 6. No primeiro passo de cada medida, o banco é populado de acordo com o modelo apresentado na Fig. 6(a). Um nó de cada tipo representado na figura é inserido no banco para cada uma das 1000 transações que serão executadas no passo posterior. No segundo passo, as 1000 transações são executadas. As transações para cada um dos cenários usam os modelos mostrados na Fig. 6(b).

No cenário *Condicional*, a i-ésima transação atualiza o atributo *attrib1* do i-ésimo nó *Person*. O cenário *Aresta Obrigatória*, devido à natureza dessa RI, realiza uma remoção de aresta ao invés de uma inserção. No caso, a aresta do i-ésimo nó *Person* para o $2i$-ésimo nó *Company* existente no grafo é removida sem violar a RI da Fig. 5. No cenário *Entrada/Saída*, a i-ésima transação cria uma aresta *owns* do i-ésimo nó *Person* para o i-ésimo nó *Company*. No cenário *Cardinalidade* o i-ésimo nó *Person* é conectado aos i, $2i$ e $3i$-ésimos nós *Company* através de arestas da classe *owns*. Nas variantes *Modificado* e *Original*, um único comando é enviado ao OrientDB por transação. Na variante *Aplicação*, isso não é possível. Logo, a aplicação realiza uma operação de *SELECT* para cada nó, de modo a validar a cardinalidade já existente de arestas, para então enviar um comando que crie as arestas entre os nós. A mesma dinâmica se repete para a variante *Aplicação* em todos os cenários, visto que é necessário validar a RI antes de efetivamente executar a operação pretendida.

A Fig. 7 mostra o tempo de execução dos quatro cenários nas três variantes mencionadas acima utilizando diagramas de caixa. As caixas estreitas com fundo branco representam os quartis centrais, totalizando 50% das medições e são dividas na mediana. Para os limites, a partir dos quais *outliers* são desenhados como pontos, foi utilizado $min(Maximo, Mediana \pm 1,5IQR)$ onde IQR é o *Inter-Quartile Range*, a altura das caixas. Além dos elementos clássicos de um diagrama de caixa foi adicionada a média, como um losango preto e o Intervalo de Confiança (IC) de 95% para a média, representado por retângulos largos. Os rótulos nas barras mostram o aumento no tempo de execução como uma proporção da variante *Original*.

No cenário *Condicional*, não há uma diferença significativa nos tempos observados entre as três variantes. Isso ocorre pois esse é o cenário onde a alteração no banco é a mais simples, consistindo em uma única operação em um nó a cada transação. Em contraste, quando são realizadas mudanças nas arestas, há impacto não apenas nas arestas, mas também nos dois nós conectados. A simplicidade da operação realizada, em termos de armazenamento, torna todo o processo de identificação de RIs a serem verificadas e a efetiva validação dessas RIs um fator mais impactante no tempo de resposta.

Nos demais cenários, observa-se que a validação de RIs diretamente no OrientDB modificado é, em média, mais eficiente do que a validação na aplicação, uma vez que os ICs para a média não intersectam. Outra importante conclusão a partir da Fig. 7 é que há uma grande intersecção entre o tempo observado com o OrientDB *Modificado* e com o OrientDB *Original*. Essa ampla intersecção impede que as diferenças observadas nas médias sejam consideradas estatisticamente significativas.

Embora a variante *Aplicação* tenha, de modo geral, apresentado resultados piores que o OrientDB *Modificado*, há casos em que essa diferença se destaca. Nos cenários de *Aresta Obrigatória* e *Cardinalidade* são necessários acessos adicionais ao BD para que a aplicação efetue a validação dos

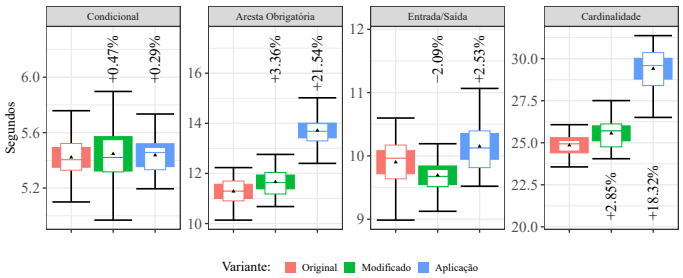


Fig. 7: Impacto da verificação de RIs no tempo de execução das transações de escrita.

dados. Uma característica do OrientDB é que, ao consultar um nó, este traz informações sobre as arestas associadas a ele. Similarmente, ao consultar uma aresta, esta traz informações sobre os nós de entrada e saída. Dessa forma, no cenário *Aresta Obrigatória* é necessário consultar a aresta existente e identificar os nós associados. Na sequência, é preciso consultar o nó de origem da aresta para verificar se este ainda possui alguma aresta que atenda à RI. No cenário *Cardinalidade*, antes de criar novas arestas é necessário consultar cada nó e verificar se o limite de arestas será respeitado. Como essas consultas trazem informações extra, elas são mais custosas para a aplicação em comparação com a variante *Modificado*, em que a validação é feita no próprio BD, o qual já possui os dados necessários para a validação.

Os experimentos executados assumem que a aplicação cliente é capaz de garantir um controle de concorrência adequado de modo a preservar a consistência das RIs, o que pode ser difícil de alcançar em cenários práticos. Duas instâncias da aplicação podem concorrentemente validar uma RI do tipo *Cardinalidade* e então concorrentemente realizar duas operações que, em conjunto, violam a RI. Em situações como esta, em que existem aplicações modificando o grafo concorrentemente, devem ser aplicadas técnicas para controle de concorrência distribuído. Como consequência, haverá uma maior complexidade da implementação, causando impactos adicionais no desempenho, além dos mostrados na Fig. 7.

V. Trabalhos Relacionados

Algumas das características dos BDGs, como serem livres de esquema (*schema-less*) e seguirem as propriedades BASE, os tornam mais flexíveis, permitindo maior agilidade e desempenho. Por outro lado, estes mesmos aspectos são responsáveis pela falta de consistência que pode ser fundamental para algumas categorias de aplicação. No entanto, cabe ressaltar que uma forte definição de esquema pode deteriorar o desempenho do SGBD. Portanto, é necessário encontrar uma solução para a falta de consistência sem perder muita flexibilidade e desempenho. A maioria dos trabalhos encontrados na literatura que tratam de RIs tem como alvo BDs relacionais. Entre os poucos que discutem o modelo de grafo, muitos comparam apenas as implementações disponíveis, enquanto outros sugerem opções simples de restrições.

O artigo de Pokorný [5] apresenta uma análise das características funcionais dos BDGs, contemplando os tópicos de armazenamento, consulta, escalabilidade, processamento de transações e uma discussão das categorias e limitações. Entre as limitações desse modelo, Pokorný lista características que não são inteiramente suportadas pelos atuais BDGs, tais como a capacidade de particionamento de dados, suporte a consultas declarativas, operações de vetor e restrições de modelo. Dentro do tópico de restrições de modelo, as RIs desempenham um papel central, embora não sejam plenamente suportadas pelos BDGs atuais.

O trabalho de Barik et al. [17] emprega BDGs para analisar possíveis caminhos de ataque de aplicações em rede. A maior parte da discussão deste artigo se concentra na análise de vulnerabilidades e ataques utilizando grafos. Na análise dos autores, as pré-condições necessárias para executar um ataque formam um grafo de dependência. Portanto, um ataque é visto como uma progressão que satisfaz sequencialmente as dependências. Os autores argumentam que o uso de RIs auxilia o processo de geração de um grafo de ataque. Por fim, é proposta uma extensão do Neo4j [20] para criar uma camada de restrição responsável pela verificação da integridade dos dados. A extensão proposta permite RIs de valores únicos de propriedades, já suportada nativamente pelo Neo4J, propriedades como chaves primárias e estrangeiras, intervalo permitido para valores de propriedades, entrada/saída (denominado modelo de aresta) e cardinalidade de aresta.

Em uma comparação entre BDs relacionais e de grafos, Pokorný [21] lista características de BDs relacionais que atualmente não são suportados pelos BDGs. Um desses recursos ausentes é a definição explícita do esquema, incluindo a especificação de RIs. Essa ausência dificulta a verificação da precisão dos BDGs. Pokorný [21] argumenta que os BDGs são baseados em um modelo lógico que contém três componentes: i) um conjunto de tipos e estruturas de dados; ii) um conjunto de operadores de inferência; iii) um conjunto de RIs. Os SGBDs de grafo atuais geralmente não possuem pelo menos um desses três componentes, sendo o suporte a RIs geralmente ausente. Posteriormente, em [16], os autores usam SGBDs de grafo para gerenciar dados fortemente conectados e consultas complexas. Reafirmam a ausência de ferramentas para garantia de consistência dos dados e citam alguns exemplos de aplicação, como redes sociais, web semântica, entre outros. Em complemento lembram que BDs de grafos são livres de *schema*, de modo que não há regras definidas para o armazenamento de dados. Segundo os autores, RIs proporcionam um mecanismo para capturar a semântica do domínio de interesse representado pelo grafo. Portanto, sugerem a definição de RIs no nível conceitual, ou nível de BD. Para isso, consideraram tipos de dados de propriedade, intervalos de valores de propriedade, disjunção de classe (um nó não pode pertencer a duas classes simultaneamente), arestas obrigatórias e valores únicos para uma composição de propriedades. O suporte às novas RIs é adicionado ao SGBD Neo4j através da extensão de sua linguagem Cypher.

O trabalho de Roy-Hubara et al. [2] discute a modelagem de dados e uma definição de esquema. Os autores apresentam uma abordagem baseada no diagrama entidade-relacionamento (ER) do domínio da aplicação e criam um mapeamento do modelo ER para um BDG junto com uma DDL. De acordo

com os autores, a abordagem poderia ser aplicada a qualquer BDG, no entanto a implementação da proposta é mencionada como trabalho futuro, o qual não foi publicado até o momento.

Por fim, Angles [7] tenta encontrar bases teóricas comuns para modelos de grafo implementados por vários BDGs. O autor adota grafos de propriedades como ponto de partida e fornece uma formalização lógica, incluindo a noção de um esquema. Essa noção básica é estendida com RIs e, em seguida, o autor propõe a sintaxe e a semântica de uma linguagem de consulta unificada. Na discussão, o autor propõe RIs para propriedades obrigatórias, arestas obrigatórias, propriedades únicas, cardinalidade de propriedades e cardinalidade de arestas. Todas elas são definidas e discutidas do ponto de vista teórico, sem discutir seu suporte nos BDGs existentes.

Em comparação aos trabalhos citados, nossa proposta contribui com a adição de novas RIs à implementação de um BDG assim como com o desenvolvimento do gerenciador de restrições, não encontrado nas demais propostas. Este gerente centraliza a manutenção e verificação dos dados além de facilitar o desenvolvimento de suporte a outras RIs de acordo com a necessidade do usuário. A Tabela I mostra quais RIs são suportadas nativamente pelo Neo4j e quais são adicionadas ao Neo4j pelas extensões propostas em [17] e [16]. No caso do OrientDB, a tabela mostra primeiramente as RIs suportadas originalmente pelo BDG e em seguida as adicionadas por este trabalho. Na tabela, as marcas de verificação mais claras representam as RIs suportadas originalmente pelo BDG, enquanto as marcas mais escuras indicam as novas RIs propostas pelos documentos mencionados no cabeçalho da tabela. Aquelas que são implementadas nativamente, mas também são repetidas com marcas de seleção mais escuras, foram reimplementadas ou aprimoradas pela proposta referenciada.

Algumas das RIs que aparecem na Tabela I requerem uma discussão mais aprofundada. Em BDs relacionais, chaves estrangeiras são usadas para representar relacionamentos entre tuplas de tabelas distintas. Em um BDG, isso pode ser considerado um *anti-pattern*, pois, em vez de usar uma propriedade de chave estrangeira, deve-se usar arestas para representar relacionamentos entre os nós. A restrição de chave primária, mencionada por alguns autores, pode ser substituída pela definição simultânea de RIs de unicidade e obrigatoriedade. Se chaves primárias são usadas para manter a integridade referencial, o usuário cai no mesmo *anti-pattern* de restrições de chave estrangeira.

Por sua vez, as RIs de disjunção de classe não permitem a associação de um único nó a duas ou mais classes. No caso do OrientDB, essa RI é uma decisão de projeto do próprio SGBD e cada nó deve pertencer a uma única classe de nó. Portanto, esta RI não se aplica ao OrientDB em seu sentido literal. Se uma única classificação extra for permitida, é possível usar uma única propriedade e limitar seu intervalo de valores usando restrições de valores máximos/mínimos ou usando a RI Expressão Regular (RegEx). Uma RegEx é uma sequência de caracteres que descreve de forma compacta um conjunto de valores permitidos. Expressões regulares podem ser usadas para descrever também conjuntos de valores permitidos, como a expressão `adult|minor`, que aceita os dois valores possíveis para uma propriedade.

TABELA I
COMPARAÇÃO ENTRE RIS NATIVAS E EXTENSÕES.

Tipos de Restrição	Neo4J	[17]	[16]	OrientDB Original	OrientDB Modificado
Aresta Obrigatória	✓	✓	✓		✓
Cardinalidade de Arestas		✓			✓
Chave Estrangeira		✓			
Chave Primária	✓	✓	✓	✓ (1)	✓ (1)
Coexistência de Rótulos		✓	✓	✓ (2)	✓ (2)
Condicional					✓
Entrada/Saída		✓			✓
Intervalo de Valor (max/min)		✓	✓	✓	✓
Propriedade Obrigatória	✓	✓	✓	✓	✓
Propriedade Única	✓	✓	✓	✓	✓
Propriedade Única Composta			✓	✓	✓
RegEx				✓	✓
Tipo da Propriedade			✓	✓	✓

(1): A restrição de Chave Primária pode ser criada pela composição das RIs de Propriedade Obrigatória e Propriedade Única.

(2): O OrientDB impõe um único rótulo (classe) por nó.

VI. CONCLUSÕES

Diante do grande volume de dados produzidos pelos sistemas computacionais, que resultou no fenômeno denominado *Big Data*, novas soluções de gerência e manipulação de dados precisaram ser desenvolvidas. A incapacidade das tecnologias tradicionais em cumprir requisitos de desempenho e escalabilidade exigidos por aplicações recentes, juntamente com a adoção de novos modelos de dados, foram motivações para o desenvolvimento dessas tecnologias de armazenamento. Nesse contexto, BDGs ganharam popularidade devido à facilidade de representar grandes volumes de dados e com alta conectividade. No entanto, esse tipo de BD ainda carece de mecanismos capazes de garantir a integridade dos dados armazenados. Dessa forma, esse trabalho apresenta a proposta de extensão de um BDG e adiciona suporte a quatro novas RIs: restrições condicional, entrada/saída, aresta obrigatória e cardinalidade.

A proposta de adição de novas RIs a um BDG ataca o problema da falta suporte à definição de regras de integridade e cria mecanismos que auxiliam a manutenção da consistência do dado. Esses mecanismos garantem a verificação do dado e o armazenamento da informação concordante com a regra definida. A fim de avaliar o custo de implementar o suporte a RIs e suas validações junto a um SGBD de grafo, foram comparadas a versão do servidor do OrientDB original, uma versão modificada com o suporte adicional a RIs, e um terceiro caso em que a validação dos dados ocorre na aplicação cliente do BD. Mediante as avaliações realizadas, observa-se que a versão modificada do OrientDB apresentou um pequeno aumento no tempo de execução das operações de manipulação de dados, tendo a versão original como base. No entanto, esse aumento não é grande o suficiente para ser considerado estatisticamente relevante. Mesmo com o pequeno aumento, em todos os casos a versão modificada foi consideravelmente mais rápida que a execução das validações no nível da aplicação cliente. A melhoria de desempenho da versão modificada em relação aos resultados da aplicação cliente são ainda mais relevantes pois não foi considerado o controle de concorrência no nível da aplicação, uma tarefa desafiadora e para a qual uma

solução eficiente não é trivial. Em adição, a implementação da versão modificada também simplifica o desenvolvimento de aplicações clientes, já que não precisam executar verificações dos dados os quais serão validados no próprio SGDB.

Apesar de atualmente a implementação do componente gerenciador de restrições funcionar apenas com o OrientDB, a lógica da solução proposta também pode ser adotada para adicionar suporte a novas RIs a outros BDGs. Além disso, apesar de adicionar suporte para apenas quatro RIs, a solução proposta permite a adição de outras restrições, com o objetivo de ampliar ainda mais os mecanismos que podem garantir a integridade dos BDGs.

REFERÊNCIAS

[1] A. Cravero and S. Sepulveda, "Using GORE in big data: A systematic mapping study," *IEEE Latin America Trans.*, vol. 17, no. 3, pp. 493–504, Mar. 2019. [Online]. Available: https://doi.org/10.1109/tla.2019.8863320

[2] N. Roy-Hubara, L. Rokach, B. Shapira, and P. Shoval, "Modeling graph database schema," *IT Professional*, vol. 19, no. 6, pp. 34–43, November 2017.

[3] M. Stokebraker, "SQL Databases v. NoSQL Databasesd," in *Communications of the ACM*, vol. 53, no. 4, 2010, pp. 10–11.

[4] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1:1–1:39, Feb. 2008. [Online]. Available: http://doi.acm.org/10.1145/1322432.1322433

[5] J. Pokorný, "Graph Databases: Their Power And Limitations," in *Computer Inf. Sys. and Industrial Manag.*, K. Saeed and W. Homenda, Eds. Cham: Springer Int. Publ., 2015, pp. 58–69.

[6] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *2014 IEEE Symp. on Security and Privacy*, May 2014, pp. 590–604.

[7] R. Angles, "The property graph database model," in *Proc. of the 12th Alberto Mendelzon Int. Workshop on Foundations of Data Manag., Cali, Colombia, May 21-25, 2018.*, 2018. [Online]. Available: http://ceur-ws.org/Vol-2100/paper26.pdf

[8] R. V. Vaz, J. D. Q. de Oliveira, and L. A. Ribeiro, "Duplicate management using graph database systems: A case study," in *Proc. of the XV Brazilian Symp. on Inf. Sys.*, ser. SBSI'19. New York, NY, USA: ACM, 2019, pp. 50:1–50:8. [Online]. Available: http://doi.acm.org/10.1145/3330204.3330260

[9] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, "Persisting big-data: The nosql landscape," *Inf. Sys.*, vol. 63, pp. 1 – 23, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437916303210

[10] G. C. G. van Erven, M. Holanda, and R. N. Carvalho, "Detecting evidence of fraud in the brazilian government using graph databases," in *Recent Advances in Inf. Sys. and Tech.*, Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, and S. Costanzo, Eds. Cham: Springer Int. Publ., 2017, pp. 464–473.

[11] K. de Faria Cordeiro, M. L. M. Campos, and M. R. S. Borges, "adapta: Adaptive approach to information integration in dynamic environments," *Computers in Industry*, vol. 71, pp. 88–102, 2015. [Online]. Available: https://doi.org/10.1016/j.compind.2015.03.002

[12] V. C. O. de Souza and M. V. C. dos Santos, "Maturing, consolidation and performance of nosql databases: Comparative study," in *Proc. of the Annual Conf. on Brazilian Symp. on Inf. Sys.: A Computer Socio-Technical Perspective - Volume 1*, ser. SBSI 2015. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2015, pp. 32:235–32:242.

[13] R. S. Sandhu, "On five definitions of data integrity," in *Proc. of the IFIP WG11.3 Working Conf. on Database Security VII*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publ. Co., 1994, pp. 257–267.

[14] T. L. Orient. Orientdb graph database. [Online]. Available: https://orientdb.com/

[15] DBEngines. Db-engines ranking of graph dbms. [Online]. Available: https://db-engines.com/en/ranking/graph+dbms

[16] J. Pokorný, M. Valenta, and J. Kovačič, "Integrity Constraints In Graph Databases," *Procedia Computer Science*, vol. 109, pp. 975 – 981, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050917311390

[17] M. S. Barik, C. Mazumdar, and A. Gupta, "Network vulnerability analysis using a constrained graph data model," in *Inf. Sys. Security*, I. Ray, M. S. Gaur, M. Conti, D. Sanghi, and V. Kamakoti, Eds., Cham, 2016, pp. 263–282.

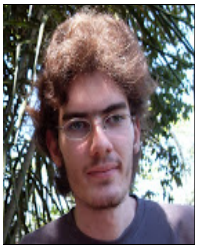
[18] H. Ip and H. Tang, "Parallel evidence combination on a SB-tree architecture," in *1996 Australian New Zealand Conf. on Intelligent Inf. Sys. Proc. ANZIIS 96*. IEEE, 1996. [Online]. Available: https://doi.org/10.1109/anziis.1996.573882

[19] F. Reina. Orient driver - experiment source code. [Online]. Available: https://bitbucket.org/fmreina/orient-driver/src/master/

[20] I. Neo4j. Neo4j graph database. [Online]. Available: https://neo4j.com/

[21] J. Pokorný, "Conceptual And Database Modelling Of Graph Databases," in *Proc. of the 20th Int. Database Engineering & App. Symp.*, ser. IDEAS '16. New York, NY, USA: ACM, 2016, pp. 370–377. [Online]. Available: http://doi.acm.org/10.1145/2938503.2938547

Fábio Reina Mestrando no Programa de Pós Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Catarina (UFSC), Brasil. Graduou em 2017 em Ciência da Computação pela UFSC e trabalha no Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) da UFSC desde 2016 com foco em Integridade em Banco de dados de Grafo e Restrições de Integridade.



Alexis Huf Doutorando no Programa de Pós Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Catarina (UFSC), Brasil. Graduou em 2016 e concluiu o Mestrado em Ciência da Computação na UFSC em 2018. Desde de 2016, trabalha no Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) da UFSC, com foco em Composição de serviços Web e tecnologias de Web Semântica.

Daniel Presser Doutorando no Programa de Pós Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Catarina (UFSC), Brasil. Graduou em Ciência da Computação pela Fundação Universidade Regional de Blumenau em 2007 e concluiu Mestrado em Ciência da Computação pela UFSC. Conduz pesquisa da área de Tolerância a Faltas, Processamento Distribuído de Grafos e Stream Processing.

Frank Siqueira Professor titular da Univ. Federal de Santa Catarina (UFSC), Brasil, na qual atua desde 2002. Graduado (1993) e Mestre (1996) em Eng. Elétrica pela UFSC, e doutor (1999) em Computação pelo Trinity College Dublin, Irlanda. É membro fundador do Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) da UFSC e pesquisador associado da Univ. Sydney, Austrália. Conduz pesquisa nas áreas de: Sistemas Distribuídos, Web Services, Web Semântica e Bancos de Dados Distribuídos.