UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JACSON LUIZ MATTE

CORRESPONDÊNCIA ENTRE ESQUEMAS ORIENTADA A AMOSTRAS DE TUPLAS

CHAPECÓ
2015

JACSON LUIZ MATTE

CORRESPONDÊNCIA ENTRE ESQUEMAS ORIENTADA A AMOSTRAS DE TUPLAS

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Denio Duarte

CHAPECÓ
2015

Luiz Matte, Jacson

Correspondência entre esquemas orientada a amostras de tuplas / por Jacson Luiz Matte. – 2015.
75 f.: il.; 30 cm.

Orientador: Denio Duarte
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal da Fronteira Sul, Chapecó, SC, 2015.

1. Correspondência entre esquemas. 2. Palavra-chave. 3. Mapeamento. I. Duarte, Denio. II. Título.

E-mail: jacsonmatte@gmail.com

JACSON LUIZ MATTE

CORRESPONDÊNCIA ENTRE ESQUEMAS ORIENTADA A AMOSTRAS DE TUPLAS

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Denio Duarte

Este trabalho de conclusão de curso foi defendido e aprovado pela banca em: ____/____/____

BANCA EXAMINADORA:

_____

Dr. Denio Duarte - UFFS

_____

Dr. Claunir Pavan - UFFS

_____

Ma. Andressa Sebben - UFFS

AGRADECIMENTOS

Primeiramente a Deus por me acompanhar, ter dado saúde, força e coragem para superar as dificuldades nessa caminhada.

Agradeço ao meu pai Lenoir e minha mãe Melânia, pelo exemplo de vida e simplicidade que foram fundamentais na construção do meu caráter. Agradeço a minha namorada Jéssica pelo apoio e incentivo durante a elaboração deste trabalho, e por ser a namorada mais amável e compreensiva possível. Agradeço a toda minha família que com muito carinho e apoio, não mediram esforços para que eu chegasse até aqui.

Agradeço ao meu orientador, Prof. Denio Duarte pelas orientações, paciência e por todo o conhecimento compartilhado para realização deste trabalho. Agradeço a Prof^{a} Andressa Sebben e ao Prof. Claunir Pavan, que colaboraram e agregaram no desenvolvimento deste trabalho, por participarem da banca de avaliação.

Obrigado a todos meus colegas e amigos que me ajudaram de alguma forma. Aos professores de modo geral que repassaram um pouco de seu conhecimento a cada etapa do curso, agregando ao meu conhecimento. Finalmente, agradeço a todas as pessoas que de forma direta ou indireta contribuíram para minha formação e acreditaram no meu potencial.

RESUMO

Correspondência entre esquemas (*Schema Matching*, em inglês) é um problema de geração de correspondência entre elementos de dois ou mais esquemas e é aplicado em muitos domínios da área de banco de dados. A tarefa de correspondência entre esquemas exige que os usuários tenham conhecimento dos esquemas dos bancos de dados envolvidos na correspondência. Porém, usuários interessados em construir sua base de dados de assuntos de seu interesse, geralmente, têm conhecimento apenas de alguns exemplos (amostras) que devem estar armazenados em um esquema no banco de dados ou em uma saída de uma consulta. Neste caso, eles acabam se deparando com a dificuldade de entender como armazenar tais dados ou como eles são representados. Neste contexto, este trabalho propõe uma ferramenta que com base em um banco de dados relacional de origem e amostras de usuário, constrói uma correspondência entre esquema, derivando automaticamente as correspondências. Essa correspondência cria um esquema em forma de visão que represente os dados digitados. A ferramenta $JMatching$ foi desenvolvida para implementar a abordagem deste trabalho: correspondência de esquemas baseado em amostras. Foram conduzidos experimentos para identificar a coerência dos esquemas e base de dados gerados. Além disso, foi medido o tempo de execução e a quantidade de memória em quatro casos de teste.

Palavras-chave: Correspondência entre esquemas. Palavra-chave. Mapeamento.

ABSTRACT

Schema mappings are high-level specifications that describe the relationship between schemas. In real-life applications schema mappings can be quite complex and, then, users must know very well schemas to be matched. However, there exist applications that users know very well the domain but details of the schemas are not known. Thus, tools that can build a target schema based on samples are demanded by users not experts in computer science but experts in the application domain. This work aims to implement an approach to schema mapping based on samples to help not experts users to build their own database. Some experiments were conducted to verify schemas and built database correcteness. Besides, we measured the amount of memory needed and execution time. In all experiments, our work showed good results.

Keywords: Schema Matching, Keywords, Mapping.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A correspondência entre esquemas é um importante problema tratado por pesquisas em banco de dados e é aplicada em várias áreas: integração entre aplicações, *data warehouses*, processamento semântico de consultas, entre outros [Rahm and Bernstein, 2001]. A tarefa de correspondência pode ser trabalhosa e longa, assim pesquisas nesta área procuram soluções para que a correspondência seja feita de forma automática e resulte em correspondências corretas.

Dados um esquema de origem *S* e um esquema de destino *T*. A correspondência entre esquemas é dada por $M = (S, T, \Sigma)$, onde Σ é um conjunto de regras que representam a correspondência dos elemento em *S* com os elementos em *T* (também chamadas de expressões de mapeamento). Segundo [Bernstein et al., 2011], um esquema é uma estrutura formal que representa um objeto, por exemplo, um esquema de uma tabela de um banco de dados. A correspondência é uma relação entre um ou mais elementos de esquemas diferentes, que representam um mesmo objeto.

Existem basicamente duas técnicas para construir Σ [Rahm and Bernstein, 2001, Shvaiko and Euzenat, 2005a, Qian et al., 2012]: as expressões de correspondência são criadas a partir dos elementos dos esquemas (correspondência baseada em elementos), e as expressões são criadas baseadas nos elementos e nos dados presentes nos esquemas (correspondência baseada em instâncias).

Em muitas aplicações, tais como integração de dados na Web, integração de esquemas, processamento de consultas, geração de novos mapeamentos de esquemas, entre outros, a correspondência entre esquemas desempenha uma função fundamental normalmente realizada pelo usuário manualmente e, às vezes, tendo uma interface gráfica para auxiliar.

Neste contexto, a correspondência de esquemas é uma tarefa importante na organização e geração de novos esquemas e, segundo [Kolaitis, 2005], tem sido um dos problemas mais complexos e mais difíceis na área de interoperabilidade de dados entre aplicações.

Geralmente, a correspondência entre esquemas é feita com auxílio de usuários conhecedores da aplicação e do banco de dados tanto no nível externo quanto no nível lógico. Isso impede que usuários especialistas na aplicação mas sem conhecimento da estrutura do esquema do banco de dados possam construir suas próprias correspondências. Desta forma, os usuários especialistas apenas na aplicação que necessitem realizar à tarefa de correspondência entre esquemas não estão preparados tecnicamente para trabalhar com as aplicações que fazem ma-

peamento entre esquemas e exigem uma compreensão da semântica de vários esquemas e suas correspondências [Qian et al., 2012].

Para atender esses tipos de usuários, alguns trabalhos tratam o problema da seguinte forma: dados uma base de dados de origem com o esquema possivelmente desconhecido pelo usuário e um conjunto de tuplas (possivelmente unitário) informado pelo usuário, é gerado um esquema de saída baseado nas duas entradas.

Um usuário especialista interessado em construir uma base de dados de assuntos de seu interesse, geralmente tem conhecimento de alguns exemplos (amostras) que devem estar armazenados em um esquema no banco de dados, ou em uma saída de uma consulta. Todavia, acaba se deparando com a dificuldade de entender como armazenar tais dados, ou como eles são representados.

Contudo, pode ser aplicado a correspondência entre esquemas a partir de uma base de dados de origem analisando sua estrutura e semântica. Porém, para que os usuários especialistas interessados consigam fazer um mapeamento customizado, faz-se necessário uma aplicação que consiga gerar esse mapeamento de forma automática.

Segundo [Hristidis and Papakonstantinou, 2002], uma grande quantidade de informações são armazenadas em bancos de dados relacionais. No entanto, a extração dessas informações exige um conhecimento do esquema que as descreve; usuários especialistas, na maioria das vezes, não tem conhecimento do esquema ou dos papéis de várias tabelas.

Assim, a área de correspondência de esquemas é bastante ampla e com várias abordagens existentes. Este trabalho tem como foco a correspondência de esquemas em nível de instância dos dados, em banco de dados relacional. Tem como característica fazer com que os dados sejam mapeados do nível externo para o nível conceitual, isolando o usuário da compreensão da semântica do esquema, relações e suas correspondências, tornando-se uma tarefa possivelmente mais fácil de ser realizada.

Para tanto, o trabalho em questão propõe, além de um estudo sobre as técnicas existentes na correspondência entre esquema, o desenvolvimento de um sistema de correspondência entre esquemas, orientado à amostra. Uma correspondência entre esquemas é a transformação de uma instância de um banco de dados de origem, para uma instância que obedece o esquema de destino. Quando se tem uma abordagem por amostra, se pressupõe a existência de um banco de dados de origem para que possa se fazer a correspondência entre as amostras e os atributos do esquema do banco de dados fonte. Como resultado, o usuário não necessita entender o esquema

de origem ou especificar os relacionamentos entre as tabelas [Qian et al., 2012].

A ferramenta proposta, chamada *JMatching*, constrói uma correspondência entre esquema, derivando automaticamente às correspondências com base em um banco de dados relacional de origem e as amostras de usuário. Essa correspondência criará um esquema em forma de visão que representará as amostras de entrada.

Foi criada uma interface, na qual usuários podem inserir as amostras. Como resultado são apresentadas uma visão dos esquemas e das consultas, gerados a partir da amostra.

O restante do trabalho está organizado da seguinte forma: o próximo capítulo aborda a correspondência entre esquemas e suas aplicações. Já no Capítulo 3, são apresentados conceitos sobre banco de dados relacional. No Capítulo 4, são apresentadas algumas abordagens de correspondência entre esquemas a nível de elemento. O Capítulo 5 apresenta a proposta de mapeamento entre esquema baseado em amostras. O Capítulo 6 apresenta a implementação da proposta (i.e., *JMatching*). No Capítulo 7, são apresentados alguns experimentos aplicados ao *JMatching*. No Capítulo 8 é apresentada a conclusão deste trabalho, bem como as experiências de desenvolvimento, resultados obtidos e algumas sugestões de trabalhos futuros.

2 CORRESPONDÊNCIA ENTRE ESQUEMAS

Segundo [Qian et al., 2012], uma correspondência entre esquemas é a transformação de uma instância de um banco de dados de origem para uma instância que obedece um esquema de destino. A Correspondência entre esquemas pode ser feita manualmente, ou pode ser obtida por meio de um sistema de mapeamento, que gera automaticamente a correspondência entre os esquemas a partir de uma representação visual [Alexe et al., 2011].

A correspondência (*match*, em inglês) é uma operação principal na manipulação de informações de esquemas que é definida como: dados dois ou mais esquemas como entrada o resultado irá produzir um mapeamento entre elementos dos esquemas que se correspondem semanticamente [Rahm and Bernstein, 2001]. Correspondência de esquemas envolve também a descoberta de uma consulta ou um conjunto de consultas que transformam os dados de origem em uma nova estrutura [Miller et al., 2000].

A correspondência é estabelecida por um conjunto de elementos mapeados, como mostrado na Figura 2.1. Os elementos da tabela $Rios$ e da tabela $Lagos$ são mapeados para a tabela de destino *Recursos de Água* (indicado por setas contínuas na figura). Perceba que os elementos de *Rios* e *Lagos* são mapeados, resultando a tabela *Recursos de Água* (indicado pela seta tracejada).

A Figura 2.1 identifica, ainda, atributos de uma correspondência individual, no qual um ou mais elementos do esquema *Rios* pode corresponder com um ou mais elementos do esquema *Lagos*, que representam os mesmos conceitos nos dois esquemas relacionais. Assim, pode obter-se as cardinalidades locais das relações entre os atributos como *1:1* (um pra um), *N:1* (muitos pra um) ou *N:N* (muitos para muitos). A cardinalidade local se aplica no elemento de correspondência, individualmente.

Por exemplo, *IDAgua* tem o mesmo papel semântico de *IdRio* e *IDLago*, compondo uma cardinalidade local de correspondência 1:1. Muitas vezes, a relação é 1:1, como *NomeRio* corresponde com *NomeLago* (Figura 2.1), mas, às vezes, pode não ser, como no exemplo de *CidadeNascente* e *UFNascente* corresponde com *Localizacao*, onde a cardinalidade local da correspondência é de *N:1*. Existe uma correspondência semântica se as restrições entre as instâncias dos esquemas são relacionadas [Bernstein et al., 2011].

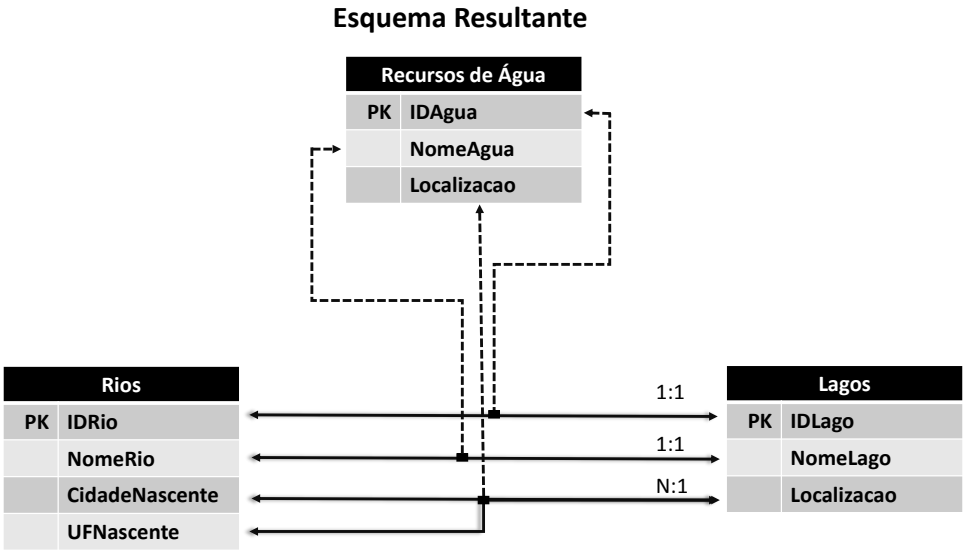

Figura 2.1: Correspondência entre esquemas e o problema de geração de correspondências que identificam elementos relacionados em dois esquemas.

2.1 Aplicabilidade

Correspondência é uma operação importante em várias aplicações tais como: em integração entre esquemas, em *data warehouses*, na integração de catálogos, *e-commerce*, na correspondência por palavras-chave, entre outras. Normalmente, estas aplicações são caracterizadas por dados estruturais / modelos conceituais e são baseadas em uma operação que leva em consideração o tempo para projetar uma correspondência, determinando, assim, o alinhamento (por exemplo, manual ou semiautomático) como um pré-requisito das aplicações que executam uma correspondência entre esquemas [Shvaiko and Euzenat, 2005b].

2.1.1 Integração entre Esquemas

A integração entre esquemas está relacionada a maioria dos trabalhos que utilizam abordagens de correspondência entre esquemas. Segundo [Batini et al., 1986], é um problema que tem sido estudado desde o início de 1980 e pode ser formulado como: dado um conjunto de esquemas construídos de forma independente têm-se o objetivo de construir uma visão global desses conjuntos.

Geralmente, os esquemas são desenvolvidos de forma independente, na sua maioria eles têm uma estrutura e nomenclatura diferentes. Isso pode acontecer quando esses esquemas são

de diferentes domínios, como um esquema imobiliário e um esquema sobre imposto de imóveis. Assim, nos domínios modelados por diferentes pessoas, todos eles terão diferentes contextos. Segundo [Rahm and Bernstein, 2001], o primeiro passo para a integração dos esquemas é identificar e caracterizar essas relações de correspondência entre esquemas: é o processo de combinação. Uma vez identificados, os elementos correspondentes podem ser unificados em um novo esquema de forma coerente. Durante esta integração, programas e consultas são criados para traduzir os dados do esquema de origem para, posteriormente, representá-los de forma integrada.

A variação do problema de integração entre esquemas é o de integrar um esquema desenvolvido de forma independente, com um determinado esquema conceitual. Novamente, isto requer conciliar a estrutura e a terminologia dos dois esquemas, o que resulta em uma correspondência entre esquemas [Rahm and Bernstein, 2001].

A Figura 2.1 exemplifica uma integração entre esquemas. A integração dos esquemas *Rios* e *Lagos* é feita através da correspondência, para gerar o esquema *Recursos de Água*. O mapeamento que realizará a integração entre os elementos desses esquemas, para um novo esquema de destino (*Recursos de Água*), envolve uma correspondência semântica entre os atributos de *Rios* e *Lagos*.

2.1.2 *Data Warehouses*

Um outro contexto do problema de integração entre esquemas, que se tornou popular na década de 90, é o da integração de fontes de dados em um *data warehouse*. *Data warehouse* é utilizado para se obter informações gerenciais de uma empresa; elas são compiladas através da extração de dados [Rahm and Bernstein, 2001]. Como mostrado em [Bernstein and Rahm, 2000], a operação de correspondência é útil para a concepção de transformações. Dada uma fonte de dados, uma abordagem para a criação de transformações apropriadas se inicia encontrando os elementos da fonte que correspondem aos dados armazenados. Esta é uma operação de correspondência entre esquemas. Como exemplo, a Figura 2.2 mostra essa correspondência: dados os esquemas de origem de *BD1* $R(a, b, z)$, de *BD2* $S(a, c, b)$ e de *BD3* $X(b, c, z)$ seus elementos são correspondidos para o esquema *RSX(a,b,zc)* (correspondência hipotética entre z e c) no *Data warehouse*. Depois que um mapeamento inicial é criado, é necessário examinar, também, a semântica de cada elemento de origem, bem como criar transformações que estão de acordo com a semântica do destino. O processo de extração requer transformar os dados do

formato de origem para o formato que será armazenado.

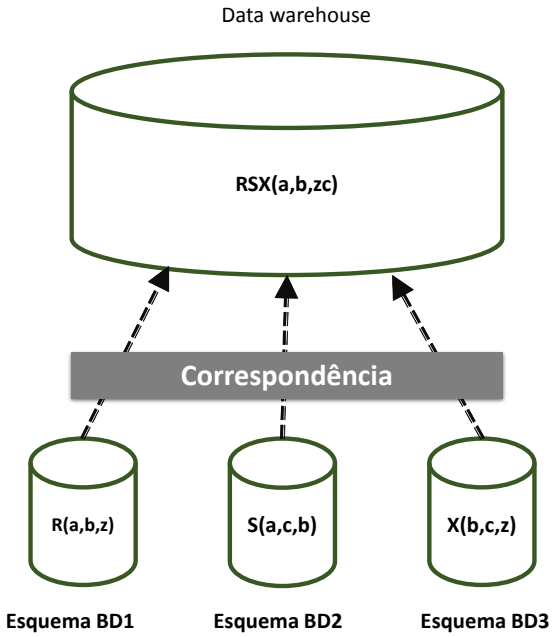

Figura 2.2: Correspondência entre esquemas em *data warehouse*.

2.1.3 Integração de Catálogos

Em aplicações *B2B* (*Business-to-Business*), os parceiros comerciais armazenam os seus produtos em catálogos eletrônicos. Os catálogos são estruturas semelhantes a árvores, conhecidas como hierarquias de conceitos com atributos. Exemplos típicos de catálogos são os diretórios de produtos do *Amazon* e do *eBay*. Uma empresa, interessada em divulgar seus produtos em mercado eletrônico (*e.g., eBay*), necessita enviar seu catálogo e, a partir dele, determinar as correspondências entre a entrada de seus catálogos e as entradas do catálogo simples do local de divulgação. Este processo de mapear entradas entre catálogos é conhecido como problema correspondência de catálogos [Bouquet et al., 2003]. Após identificadas, as correspondências entre as entradas dos catálogos são analisadas, com o intuito de gerar expressões de consulta que, automaticamente, traduzem as instâncias de dados entre os catálogos [Velegrakis et al., 2005]. Com os catálogos alinhados, os usuários têm acesso unificado aos seus produtos que estão à venda no catálogo.

2.1.4 *E-commerce*

Atualmente, o *e-commerce* vem sendo utilizado na correspondência entre esquemas com a finalidade de traduzir mensagens. Os parceiros comerciais se utilizam da troca de mensagens para descrever suas transações comerciais. Cada parceiro comercial tem seu próprio formato de mensagem, uma vez que esses formatos podem ser diferentes em sua sintaxe e em seus esquemas de mensagens [Rahm and Bernstein, 2001].

Para possibilitar que os sistemas consigam fazer a troca de mensagens, os desenvolvedores têm a tarefa de converter as mensagens para um único padrão, dentre todos os formatos produzidos pelos parceiros comerciais.

Uma parte do problema é a tradução das mensagens entre diferentes esquemas de mensagens. Esquemas de mensagens podem possuir características que os diferenciam como: nomes diferentes, tipos de dados, estrutura e outras especificações. Por exemplo, pode ser representado por uma estrutura simples, que lista os campos, enquanto o outro grupo possui mais campos relacionados; ambos podem utilizar estruturas parecidas, porém os campos podem ter diferentes combinações.

Segundo [Rahm and Bernstein, 2001], tradução entre diferentes esquemas de mensagens é, em parte, um problema de correspondência entre esquemas. Desenvolvedores de aplicativos precisam especificar manualmente como as mensagens estão relacionadas. A abordagem de correspondência entre os esquemas de mensagens poderia reduzir a quantidade de trabalho manual, auxiliando o desenvolvedor que, posteriormente, poderia validar e modificar, se necessário.

2.1.5 Correspondência por Palavras-chave

As abordagens que realizam correspondência para resolver problemas de integração entre esquemas, *Data warehouses* e *e-commerce* fornecem funções eficientes para gerar uma correspondência e tem como usuários os desenvolvedores. Os desenvolvedores possuem um conhecimento aprofundado da estrutura de vários esquemas, semânticas e suas correspondências. Essas correspondências não se tornam adequadas para usuários que não estão preparados tecnicamente que não possuem um conhecimento prévio ou que precisem realizar um grande número de correspondência entre esquemas.

Dois exemplos de abordagens propostas em ([Qian et al., 2012] e [Shen et al., 2014]) trabalham com amostras (palavras-chave) de usuários. Nessas abordagens, usuários leigos con-

seguem realizar a tarefa de correspondência entre esquemas de forma automatizada, gerando sua própria fonte de dados e extraindo um conjunto de consultas mínimas em relação a sua fonte de dados. Assim, o usuário não precisa fornecer nenhuma informação de correspondência explícita a nível de atributo, que segundo [Qian et al., 2012] isola o usuário de uma possível estrutura complexa dos esquemas de origem e de suas correspondências e não necessita nenhum domínio, em relação as operações específicas de mapeamento entre esquemas.

Nas abordagens citadas, o usuário não necessita conhecer o esquema, relações e nenhuma atribuição abaixo do nível externo de um SGBD para realizar um mapeamento e obter seu esquema correspondente. A Figura 2.3 apresenta um exemplo da correspondência por palavras-chave. No exemplo, o usuário digitou as palavras $Carlos$, *Computação* e *Banco de Dados* que são utilizadas para propor um esquema baseado em um esquema de um banco de dados fonte (nesse caso um banco de dados acadêmico). Assim, por meio de uma correspondência, *Carlos* é encontrado no atributo *nome* da tabela *Aluno*, o mesmo para palavra *Computação*, encontrada no atributo *nome* da tabela *Cursos* e a palavra *Banco de Dados*, encontrada no atributo *nome* da tabela *Disciplina* (correspondência indicada pelas setas tracejadas largas). A posteriori, são verificados seus relacionamentos e o esquema *ESQ* é retornado para o usuário em uma forma de visão. Uma segunda visão é apresentada com as consultas *Q1* e *Q2* geradas em relação ao esquema *ESQ* e as entradas do usuário. As consultas são extraídas das tabelas que possuem chave-estrangeira *Matriculas* e *AlunoDisciplina*; a seta tracejada indica a relação entre as tabelas e as consultas. E os números *1*, *2* e *3* mostrado na visão das consultas indicam em qual coluna da tabela de amostras estão relacionadas. Essas abordagens utilizam-se da eliminação de candidatos inválidos para gerar as visões corretas, sendo este o principal desafio. As visões, extraídas através destas abordagens, podem auxiliar os usuários leigos a construir sua própria base de dados e obter suas consultas. O Capítulo 4 apresenta de forma mais detalhada as abordagens descritas nesta seção.

2.2 A Operação de correspondência

A escolha de uma representação de correspondência é o passo inicial para implementar a correspondência entre esquemas. Existem diversas abordagens que realizam a correspondência, porém elas dependem muito das informações dos esquemas que são utilizados e de suas

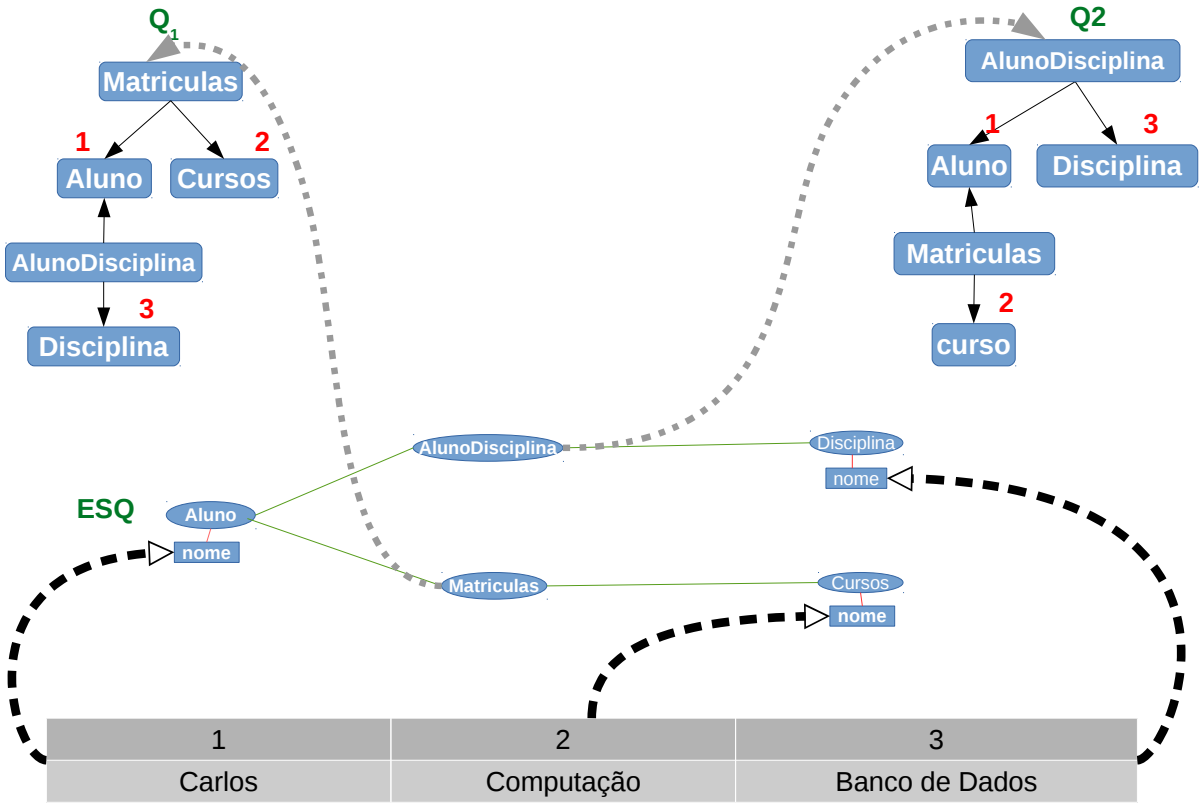

Figura 2.3: Correspondência através de amostras na geração de visões.

interpretações [Rahm and Bernstein, 2001].

Uma representação de esquema que melhor se adapte a uma determinada abordagem deve ser escolhida. Por exemplo, pode-se utilizar a representação entidade-relacionamento (*ER*) ou orientada a objetos (*OO*) para construir uma determinada correspondência [Rahm and Bernstein, 2001].

Algumas abordagens, que implementam a correspondência entre esquemas, são semelhantes a uma junção (*join*) em banco de dados relacional. Tanto a correspondência quanto a junção, que são operações binárias, determinam pares de elementos correspondentes a sua operação. Todavia, existem algumas diferenças. Correspondência atua sobre metadados (elementos de esquemas) e junções em dados (tuplas de tabelas). Além disso, a correspondência é mais complexa que a junção. Cada elemento contido no resultado de uma junção combina apenas um elemento da primeira entrada com um elemento correspondente da segunda entrada. Já um elemento resultante de uma operação de correspondência pode relacionar vários elementos, de ambas as entradas [Rahm and Bernstein, 2001].

Para tanto, a semântica que envolve uma junção é especificada por uma expressão de

comparação simples, por exemplo, uma condição de igualdade para uma junção natural (*natural join*) que deve suportar todos os elementos de entrada correspondente. De outra maneira, cada elemento em um resultado de uma correspondência pode ter uma expressão de correspondência diferente, por exemplo, uma igualdade, adição ou concatenação. Desta forma, a semântica da operação de correspondência é menos restrita para realizar suas operações do que a junção e é mais difícil de obter por meio de uma forma mais compacta [Rahm and Bernstein, 2001].

A correspondência é composta por um conjunto de elementos mapeados que é a união de todos os elementos correspondidos entre dois ou mais esquemas, por exemplo, os elementos de *Rios* e *Lagos* (Figura 2.1) podem ser correspondidos entre si. Além disso, cada elemento do mapeamento pode ter uma expressão de correspondência que especifica seus relacionamentos.

Por exemplo, a Figura 2.1 mostra três esquemas: *Rios*, *Lagos* e *Recursos de Água*. Uma correspondência entre *Rios* e *Lagos* pode conter um mapeamento simples, como *Rios.IDRio* para *Lagos.IDLago* com a expressão *Rios.IDRio = Lagos.IDLago*. Um elemento pode ser mapeado com uma expressão de concatenação "(*Rios.CidadeNascente, Rios.UFNascente*) = *Lagos.Localizacao*", que utiliza dois elementos vindos de atributos do esquema $Rios$, para descrever a correspondência com um outro atributo do esquema *Lagos*.

Os critérios utilizados para combinar *Rios*, *Lagos* e *Recursos de Água* são baseados em heurísticas, uma vez que não é possível encontrar um modelo matemático para tal. Mesmo não sendo totalmente satisfatório produzir um mapeamento por heurística, por meio deste se obtém um resultado em que o usuário pode considerar uma boa correspondência.

2.3 Técnicas de Correspondência entre Esquemas

A Figura 2.4 apresenta uma classificação proposta por [Rahm and Bernstein, 2001] das técnicas existentes na correspondência entre esquemas. Nota-se que as técnicas de mapeamento são divididas em dois grupos: um grupo que pertence às abordagens de correspondências individuais e outro grupo das abordagens combinadas. Segundo [Rahm and Bernstein, 2001], nas abordagens individuais cada ferramenta calcula uma correspondência, baseada em um único critério de correspondência; nas abordagens combinadas, cada ferramenta utiliza-se de mais de uma correspondência individual e vários critérios.

As abordagens de correspondências individuais se dividem em outros dois grupos, conforme a Figura 2.4. Um grupo é baseado somente em esquemas e usa apenas as informações do esquema; e o outro grupo é baseado em conteúdo/instância e considera apenas os dados da

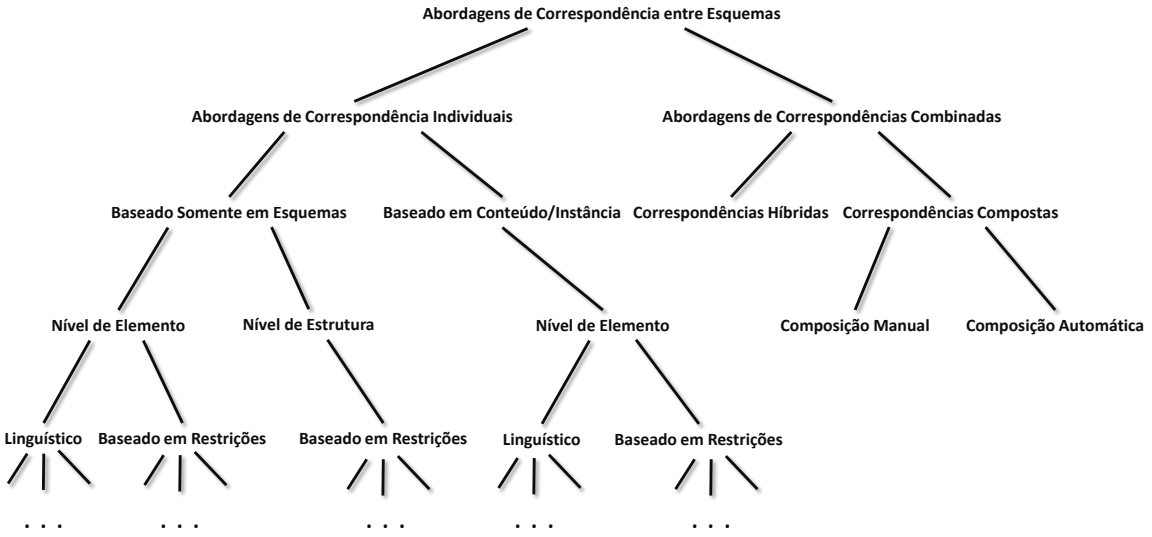

Figura 2.4: Classificação das técnicas de correspondência entre esquemas [Rahm and Bernstein, 2001].

instância. O grupo baseado em esquemas está relacionado ao nível de elemento, usando como critério os atributos, e ao nível de estrutura, usando como critério a combinação dos elementos que, por sua vez, possuem as ferramentas de correspondência por linguística e por restrições. O grupo baseado em conteúdo/instância está relacionados a nível de elemento e possui as ferramentas de correspondência por linguística e por restrições, que será detalhando na Seção 2.3.1, pois é a técnica que será utilizada como base para o trabalho.

Correspondência por linguística usa a abordagem que pode basear-se em nomes e textos, ou seja, palavras ou frases, para encontrar elementos de esquema similares, semanticamente [Bernstein et al., 2011]. Por exemplo: correspondência por nome que combina elementos do esquema com nomenclaturas iguais ou semelhantes e correspondência por descrição, no qual é avaliado, semanticamente, os comentários deixados na estrutura do esquema; por meio desses comentários pode-se determinar as similaridades.

Correspondência por restrições usa como base tipos de dados, intervalos de valores, unicidade, nulidade e chaves estrangeiras [Bernstein et al., 2011]. Se ambos os esquemas de entrada contêm tais informações, elas podem ser usadas por uma correspondência, para determinar a similaridade dos elementos do esquema [JAMES and SHAMKANT].

As abordagens de correspondências combinadas se dividem em dois grupos, conforme a Figura 2.4. Um grupo é formado pelas correspondências híbridas e outro grupo é formado

pelas correspondências compostas.

As correspondências híbridas são a combinação de vários resultados de correspondência produzidos por diferentes algoritimos [Shvaiko and Euzenat, 2005b]; por exemplo Cupid [Madhavan et al., 2001] implementou um algoritimo de correspondência híbrida, compreendendo técnicas de correspondência entre esquemas por linguística e estruturais e, também, calculou coeficientes de similaridade com ajuda de um dicionário de sinônimos.

As correspondências compostas se dividem em uma composição manual em que o resultado da correspondência é obtida pela interação do usuário, por meio da ferramenta de correspondência; é provável que o usuário tenha um grande número de interações e uma composição automática, que reduz o número de interações em relação à manual e pode ser feita pela própria implementação de correspondência.

2.3.1 As Abordagens de Correspondência Baseadas em Conteúdo/Instância

Dados em nível de instância/conteúdo podem revelar importantes informações sobre o conteúdo e semântica dos elementos do esquema. Isto é especialmente verdadeiro quando a informação útil de um esquema é limitada, como é nos casos de dados semiestruturados. Em um caso extremo o esquema não é conhecido mas um esquema pode ser construído a partir de dados de instância, criado manualmente ou automaticamente (*e.g.*, um "guia de dados" [Goldman and Widom, 1997], um grafo do esquema, palavras-chave fornecidas por usuário). Mesmo quando contém informações importantes do esquema, o uso de correspondência em nível de instância pode ser útil para descobrir interpretações incorretas sobre informações do esquema. Por exemplo, ela pode ajudar a resolver uma ambiguidade de possíveis igualdades que é tratada pela correspondência em nível de esquema, selecionando os elementos para a correspondência em que as instâncias são mais semelhantes eliminando as possíveis ambiguidades [Rahm and Bernstein, 2001].

Este capítulo apresentou os conceitos de correspondência entre esquemas. Tratou das abordagens individuais e combinadas, apresentando uma breve explicação sobre as características e aplicações de cada uma delas. Dentre as técnicas apresentadas, este trabalho vai se apoiar na abordagem de correspondência individual baseada em instância/conteúdo a nível de elemento, utilizando abordagens de correspondências por linguísticas e por restrições. Essas abordagens serão aplicadas em um banco de dados relacional cujo conceitos serão apresentados no próximo capítulo.

3 BANCO DE DADOS RELACIONAL

O modelo de dados relacional foi proposto por *Edgar Frank Codd*, em 1970 [Codd, 1970], para substituir os modelos de dados hierárquicos e o modelo de rede, propostos na década de 1960 [Ramakrishnan and Gehrke, 2008]. O modelo de dados relacional possui uma representação simples dos dados e facilita a construção de consultas complexas. Atualmente, a grande maioria dos *SGBDs* é baseado no modelo relacional. Por exemplo, os sistemas da família *DB2* da *IBM* como o *Oracle* e *SQLServer* e outros de código aberto como *MySQL* e *PostgreSQL* [Elmasri and Navathe, 2011].

Este capítulo apresenta, brevemente, alguns conceitos de banco de dados relacional, bem como suas características, restrições, linguagem de consulta e representação dos metadados. Esses conceitos são importantes para o entendimento da proposta deste trabalho.

3.1 Conceitos do Modelo Relacional

Um modelo de dados é uma coleção de ferramentas conceituais que descreve os dados, relações entre dados, semântica de dados e restrições de consistência. Existem vários modelos de dados para representar um banco de dados, porém o mais utilizado atualmente é o modelo relacional. Os bancos de dados que se apoiam no modelo relacional são chamados de banco de dados relacionais. Banco de dados relacionais consideram seu conjunto de dados como relações. Uma relação representa uma tabela que é composta por linhas (ou tuplas) e colunas (ou atributos) [Silberschatz et al., 2006].

Um banco de dados relacional é caracterizado por possuir um esquema que descreve as tabelas. Um esquema de banco de dados é formado pelo nome e atributos das tabelas do banco de dados [Ramakrishnan and Gehrke, 2008]. A Figura 3.1 apresenta um esquema de um banco de dados relacional, que modela uma aplicação de um sistema acadêmico. Para construir uma tabela é necessário definir nomes, domínios e restrições dos atributos ou colunas que compõem a tabela [Silberschatz et al., 2006].

Cada atributo de uma tabela é o nome de um papel desempenhado por algum domínio específico. O domínio restringe o conteúdo que pode ser associado a cada atributo da tabela [Silberschatz et al., 2006]. Por exemplo, Na Figura 3.1, a Tabela *Matriculas* possui o atributo *idAluno* representando o tipo de domínio *int*, podendo apenas receber valores inteiros.

Restrições são derivadas das regras de um esquema que o banco de dados representa e

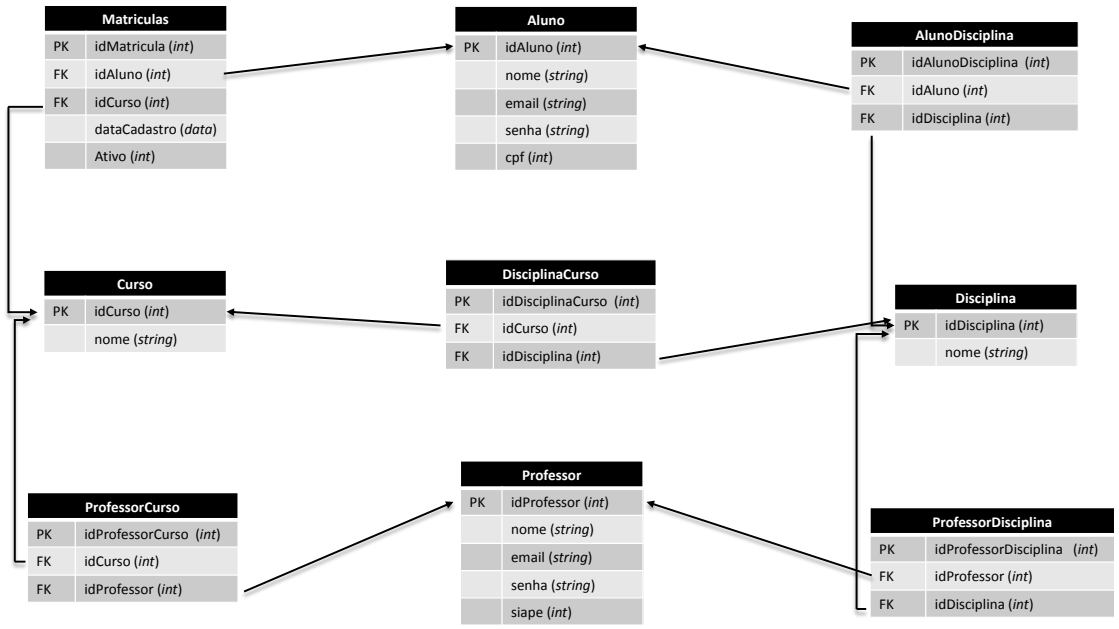

Figura 3.1: Modelo lógico de um sistema acadêmico.

definem alguma restrição para o armazenamento e manutenção dos dados. O banco de dados que respeita as restrições impostas em um determinado estado está em um estado válido [Elmasri and Navathe, 2011]. Os principais tipos de restrições são:

- Restrições implícitas (modeladas no próprio modelo de dados);
- Restrições explícitas (expressas no próprio esquema de relação);
- Regras de negócio (restrições implementadas pela aplicação);
- Dependências funcionais (restrições de dependência de dados).

Restrições baseadas em esquemas são divididas em cinco tipos de restrições. As restrições de domínio, restrições de chaves, restrições sobre valores nulos, restrições de integridade de entidade e restrições de integridade referencial. O foco deste trabalho está na restrição de chave primária e restrição de chave estrangeira.

A Figura 3.2 representa instâncias de algumas tabelas do esquema do banco de dados do sistema acadêmico da Figura 3.1.

A instância de uma chave primária não pode possuir valores repetidos [Elmasri and Navathe, 2011]. Por exemplo, não seria possível utilizar o atributo *nome* da tabela *Aluno* como

Matriculas

idMatricula	idAluno	idCurso	dataCadastro	Ativo
100	200	10	07/07/2010	1
105	235	11	08/03/2011	1
110	230	10	10/03/2010	1

Aluno

idAluno	nome	email	senha	cpf
200	Jacson Matte	jacson.matte@gmail.com	23%(&*fvD	23568178965
230	Carlos Carniel	c.carlos@gmail.com	24!@*!)gkL	215786318786
235	Jéssica Senger	j.senger@gmail.com	)(*@jlm543	52455698411

Curso

idCurso	nome
10	Ciência da Computação
11	Administração
19	Matemática

Figura 3.2: Instância do sistema acadêmico.

chave primária, uma vez que pode existir mais de um aluno com o mesmo nome. Neste exemplo, o campo escolhido foi *idAluno*, sendo que seus valores não se repetem. Na Figura 3.1 o modificador *PK* indica que o atributo é uma chave primária.

Uma chave estrangeira é um atributo (ou a combinação de atributos) usado para fazer um ligação entre os dados em duas tabelas, quando o atributo que contém o valor da chave primária é referenciado por um atributo de outra tabela. As chaves estrangeiras também podem assumir valores nulos [Silberschatz et al., 2006]. Por exemplo, na Figura 3.2, para assegurar que apenas os alunos e cursos válidos possam estar na tabela *Matricula*, foram definidas duas chaves estrangeiras *idAluno* e *idCurso* na tabela *Matriculas* (veja Figura 3.1). Qualquer valor que apareça no atributo *idAluno* da tabela *Alunos* pode aparecer no atributo *idAluno*, da tabela *Matricula*. Utilizando o mesmo raciocínio, qualquer valor no atributo *idAluno*, da tabela *Matricula*, deve existir no atributo *idAluno*, da tabela *Aluno*, ou ser nulo (relação indicado pelas setas na Figura 3.1). O mesmo critério se aplica para os atributos *idCurso* das tabelas *Cursos* e *Matricula*: o aluno **Jacson Matte**, com identificador *200*, está matriculado no curso **Ciência da Computação**, com identificador 10, pois os identificadores existem na tabelas *Aluno* e *Curso*. Os atributos *idCurso* e *idAluno* são chamados de chaves estrangeiras. Na Figura 3.1 o modificador *FK* indica que o atributo é uma chave estrangeira. Não necessariamente os atributos relacionados na definição de uma chave estrangeira devem possuir o mesmo nome, mas devem

possuir o mesmo domínio.

Além da necessidade de encontrar uma forma de definir esquemas para os banco de dados, também é necessário existir uma forma de manipular os dados. Uma das formas de se manipular os dados em uma instância de um modelo relacional é a álgebra relacional. Álgebra relacional é uma maneira teórica de manipular o banco de dados relacional. Está dividida em seis operações fundamentais: seleção, projeção, produto cartesiano, renomear, união e diferença de conjuntos [Ramakrishnan and Gehrke, 2008]. O trabalho tem como norte as operações de seleção, projeção e junção, pois estas operações serão suficientes na elaboração do algoritmo proposto sobre mapeamento entre esquemas orientado à amostra.

A seleção seleciona tuplas que satisfazem à condição de seleção. É usado a letra grega sigma (σ) para caracterizar a seleção. Sua sintaxe é $\sigma_{<predicado>}(tabela)$ [Silberschatz et al., 2006]. Predicado é o conjunto de expressões lógicas que satisfazem alguma linha da tabela. Por exemplo, na Figura 3.2, considerando a tabela $Aluno$, para selecionar uma tupla cujo o valor do atributo *nome* seja *Carlos Carniel*, escreve-se:

$$\sigma_{\ nome = \text{" Carlos Carniel "}}\ \text{(Aluno)}$$

Um predicado pode ser representado por um conjunto de condições, conectados pelos operadores lógicos: conectivo binário (E $\wedge$, OU $\vee$) e o conectivo unário (negação ($\neg$)) [Ramakrishnan and Gehrke, 2008]. Por exemplo, a partir da tabela *Matriculas*, para selecionar o curso com identificador maior que *10* e o aluno com o identificador maior que *200*, escreve-se:

$$\sigma_{\ idCurso > 10 \wedge idAluno > 200}\ \text{(Matriculas)}$$

A tabela resultante dessa operação é apresentada na Figura 3.3.

Matriculas

idMatricula (PK)	idAluno (FK)	idCurso (FK)	dataCadastro	Ativo
105	235	11	08/03/2011	1

Figura 3.3: Resultado da seleção combinando os predicados.

A projeção, indicada pela letra grega pi (π), é uma operação que filtra as colunas de uma tabela. Através da projeção pode-se retornar um subconjunto dos atributos do resultado de uma consulta. A sintaxe da projeção é $\pi_{<atributos>}(tabela)$. Por exemplo, usando a instância *Aluno*, da Figura 3.2, para listar apenas o nome e o *e-mail* de todos os alunos, escreve-se:

$$\pi_{\ nome,\ email}\ \text{(Aluno)}$$

A tabela resultante dessa operação é apresentada na Figura 3.4.

A operação de junção é representada pelo símbolo $\bowtie$. A junção faz uma comparação entre atributos de duas tabelas, por meio de um predicado. O resultado da junção é um conjunto que forma uma nova tabela, correspondente ao predicado. Sua sintaxe é $< tabela1 > \bowtie_{<predicado>} < tabela2 >$ [Ramakrishnan and Gehrke, 2008]. Por exemplo, usando as instância da Figura 3.2, deseja-se relacionar a tabela *Curso* com a tabela *Matriculas* para saber quais cursos já possuem matrículas, escreve-se:

$$\text{Curso} \bowtie_{curso.idCurso = Matriculas.idCurso} \text{Matriculas}$$

nome	email
Jacson Matte	jacson.matte@gmail.com
Carlos Carniel	c.carlos@gmail.com
Jéssica Senger	j.senger@gmail.com

Figura 3.4: Resultado da projeção da tabela Aluno.

A tabela resultante dessa operação é apresentada na Figura 3.5.

idCurso (PK)	nome	idMatricula (PK)	idAluno (FK)	idCurso (FK)	dataCadastro	Ativo
10	Ciência da Computação	100	200	10	07/07/2010	1
10	Ciência da Computação	110	230	10	10/03/2010	1
11	Administração	105	235	11	08/03/2011	1

Figura 3.5: Resultado da junção.

Essas operações de álgebra relacional são necessárias para emitir consultas. Todas as operações apresentadas podem ser implementadas em *SQL* (*Structured Query Language*) [Ramakrishnan and Gehrke, 2008].

Vale ressaltar que um sistema gerenciador de banco de dados não armazena informações do esquema junto com os dados. São criadas estruturas próprias para descrever os esquemas. Essa estrutura, representada por um conjunto de tabelas, é chamada de catálogo ou dicionário de dados. Os dados armazenados nos catálogos são chamados de metadados. A próxima seção apresenta como são organizados os metadados de um esquema em um banco de dados relacional.

3.2 Metadados

Metadados representam "dados sobre dados" [Ramakrishnan and Gehrke, 2008]. Metadados são informações armazenadas em estruturas conhecidas como catálogos ou dicionário de

dados. O catálogo contém informações como a estrutura de cada arquivo, o tipo e o formato de armazenamento de cada item de dado e suas restrições. Essas informações de um esquema são armazenadas no catálogo de um sistema de banco de dados relacional. As informações do catálogo podem ser usadas pelo *SGBD* ou por usuários que precisam da informação sobre o esquema [Silberschatz et al., 2006].

A Figura 3.6 ilustra, parcialmente, o esquema de um catálogo que representaria o sistema acadêmico da Figura 3.1. O usuário consegue ler o conteúdo das tabelas do catálogo mas não pode apagar, inserir ou atualizar dados a partir dele. Quem faz todas as manipulações sobre os dados do catálogo é o SGBD [Elmasri and Navathe, 2011].

dicTable	
PK	idDicTable (*int*)
	nomeEsquema (*string*)
	nomeTabela (*string*)
	linhasTabela (*int*)
	dataCriacao (*data*)

dicColumns	
PK	idDicColumns (*int*)
	nomeEsquema (*string*)
	nomeTabela (*string*)
	nomeColuna (*string*)
	tipoDado (*string*)

Figura 3.6: Parte de um modelo lógico que representa um sistema acadêmico.

dicTable

idDicTable (PK)	nomeEsquema	nomeTabela	linhasTabela	dataCriacao
1	Acadêmico	Matriculas	3	02/03/2010
2	Acadêmico	Aluno	3	02/03/2010
3	Acadêmico	Curso	3	02/03/2010

dicColumns

idDicColumns (PK)	nomeEsquema	nomeTabela	nomeColuna	tipoDado
1	Acadêmico	Matriculas	idMatricula	int
2	Acadêmico	Matriculas	idAluno	int
3	Acadêmico	Matriculas	idCurso	Int
4	Acadêmico	Matriculas	dataCadastro	Data
5	Acadêmico	Matriculas	ativo	Int
6	Acadêmico	Disciplina	idDisciplina	Int
7	Acadêmico	Disciplina	nome	String
...	...	...	...	...

Figura 3.7: Instância do catálogo do sistema acadêmico.

O exemplo da Figura 3.6 apresenta partes do modelo lógico de um catálogo sobre um sistema acadêmico. A Figura 3.7 apresenta a parte das instâncias do catálogo. Observa-se, na Figura 3.6, que não existe diferença na estruturação entre o modelo lógico do catálogo e

do sistema acadêmico, representado na Figura 3.1. São definidos atributos para representar as características do esquema, como o atributo *linhaTabela*, que representa a quantidade de linhas na tabela.

Já na Figura 3.7, os dados são inseridos na tabela *dicTable* e *dicColumns* do catálogo pelo próprio SGBD. Por exemplo, no atributo *nomeColuna*, da tabela *dicColumns*, o dado *idMatricula* foi inserido. Este dado representa o nome de uma coluna do modelo lógico da Figura 3.1, bem como outras informações de como o domínio do dado é representado e atribuído ao atributo *tipoDado* da tabela, na Figura 3.7.

Nota-se que o banco de dados armazena, de forma detalhada, as informações de um esquema no catálogo, auxiliando usuários que precisam dessas informações. Neste trabalho, será utilizado o catálogo de um banco de dados relacional para montar uma estrutura que servirá de auxílio para o algoritmo proposto sobre correspondência entre esquemas, baseado em amostras.

4 ABORDAGENS DE CORRESPONDÊNCIA ENTRE ESQUEMAS POR AMOSTRAS

Foram selecionados três trabalhos que possuem objetivos parecidos com este [Qian et al., 2012, Shen et al., 2014, Hristidis and Papakonstantinou, 2002] e que serviram de base para elaboração desta proposta. Em [Qian et al., 2012] foi proposta a ferramenta chamada *MWEAVER*, que realiza a correspondência entre esquemas somente com amostras de usuário e um banco de dados fonte, para resultar em um novo banco de destino. Em [Hristidis and Papakonstantinou, 2002] foi proposta a ferramenta chamada *DISCOVER* que, somente com palavras-chave fornecidas pelo usuário, consegue gerar um conjunto de consultas válidas através da correspondência entre esquemas. Já [Shen et al., 2014], baseado na abordagem proposta em [Hristidis and Papakonstantinou, 2002], propôs uma nova ferramenta chamada *FILTER* que por meio de amostras de usuário, retorna o conjunto de consultas mínimas utilizando de técnicas de correspondência entre esquemas. Percebe-se que todas as ferramentas citadas usam abordagens de correspondência entre esquemas, em nível de elemento, com amostras ou palavras-chave fornecidas pelo usuário.

Nas próximas seções são descritas as técnicas de correspondência entre esquemas implementadas nas ferramentas *MWEAVER*, *DISCOVER* e *FILTER*, utilizando uma metodologia de apresentação que segue os seguintes passos: (i) a descrição de cada ferramenta, (ii) a apresentação de um pseudo-código que implementa as técnicas da ferramenta, e (iii) um exemplo do funcionamento da ferramenta, além das características que as fazem diferir da proposta deste trabalho. A Figura 4.1 apresenta um esquema e sua instância de um banco de dados acadêmico simplificado. Esse banco de dados é utilizado como entrada para exemplificar o funcionamento das ferramentas estudadas.

4.1 *MWEAVER*

Em [Qian et al., 2012] foi proposta uma ferramenta de mapeamento entre esquema, orientado à amostra, chamado *MWEAVER*. A ferramenta constrói, automaticamente, o mapeamento entre esquemas, a partir de instâncias fornecidas pelo usuário. Com isso, o usuário não precisa fornecer nenhuma informação de correspondência em nível de atributo, isolando o usuário de um entendimento complexo sobre o esquema de origem e o mapeamento de destino. O usuário somente entra com amostras em uma $interface$, para construir sua própria base de

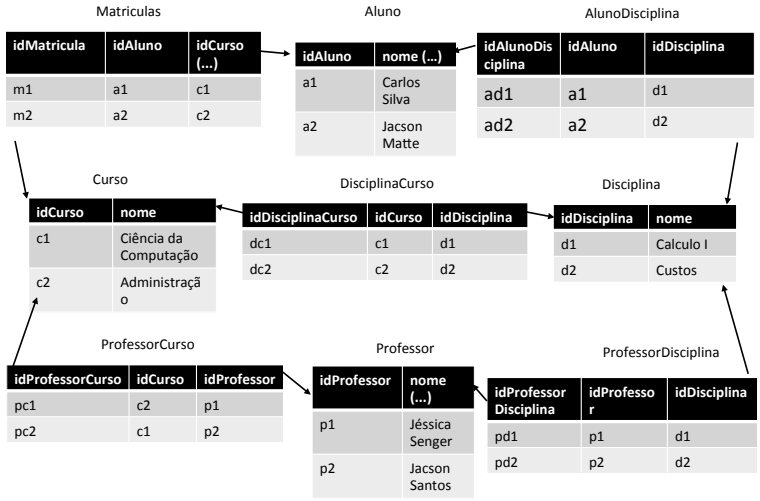

Figura 4.1: Modelo lógico simplificado e instância de um sistema acadêmico.

dados.

A ferramenta *MWEAVER* é composta pelo algoritmo de correspondência entre esquemas, chamado *TPW* (*tuple path weaving*). Seu funcionamento está dividido em cinco etapas principais: buscar ocorrências das amostras; gerar caminho de mapeamentos pares; gerar o caminho de tuplas pares; construir o caminho de tuplas completo e, classificar os mapeamentos. A Figura 4.2 apresenta as quatro primeiras etapas do processo. O Algoritmo 1 representa todas as etapas para geração dos mapeamentos candidatos.

Entrada : Base de dados D, amostras de usuário $(E_1,...,E_n)$ e grafo G do esquema ;
Saída : Mapeamento candidato M;
MapaLocalizacao L, MapeamentoCandidato M;
$L \leftarrow$ CriaMapaLocalizacao(D, E_s);
CamMapPar $\leftarrow$ CriaCamMapPar(L, G);
foreach *CMP in CamMapPar* **do**
  if *CMP NOT IN CamTuplePar and SQL(CMP)* $!=$ *"empty"* **then**
    CamTuplePar $+=$ CMP;
  end
end
$M \leftarrow$ CamTuplaPar$_1$;
foreach *CTP in CamTuplePar* **do**
  if ! *weave(*M, $CTP + 1$*)* **then**
    return -1;
  end
end
ranking(M);

Algorithm 1: *TPW*

4.1.1 Buscar ocorrências das amostras

Na primeira etapa do Algoritmo 1, busca-se os atributos no banco de dados de origem que contém pelo menos uma amostra. As entradas para essa etapa realizada pela função *CriaMapaLocalização* (linha 4) são a base de dados D e as entradas de usuário E_s que retorna o mapa de localização L.

Por exemplo, na Figura 4.2 que considera como entrada o banco de dados da Figura 4.1, o usuário digitou os seguintes dados de entrada: *Carlos da Silva*, *Ciência da Computação* e *Banco de Dados I* (Figura 4.2 (a)). Após isso, procura-se primeiro pela amostra *Carlos da Silva*, no banco de dados de origem D, o resultado seria L(1) = {*Aluno.nome, Professor.nome*}, pois esses são os atributos que contém *Carlos da Silva*. Da mesma forma são criados todos os L_s, como mostrado na Figura 4.2 (b). L é chamado de *mapa de localização*.

4.1.2 Geração de caminho de mapeamentos pares

Os caminhos de mapeamento são gerados pela função *CriaCamMapPar* (Linha 5) que recebe como parâmetro o *mapa de localização* L e o grafo dirigido G (para o estudo de caso, G representa o modelo lógico da Figura 4.1), onde os nodos representam as tabelas e os vértices as relações de chave-estrangeira. Os caminhos de mapeamento pares representam o caminho entre os itens do mapa de localização L no grafo G.

Por exemplo, dados dois elementos extraídos do mapa de localização *L*: *Aluno.nome* e *Curso.nome*. Um caminho de mapeamento são as tabelas que contém os elementos e possuem uma ligação no grafo do esquema G. Neste exemplo, essa relação é entre as tabelas $Aluno$ e $Curso$, sendo que deve existir um caminho no grafo do esquema que ligue as tabelas, que é representado pela tabela *Matriculas*. O caminho no grafo também é delimitado pela quantidade de junções entre as tabelas. Duas junções são o suficiente para representar o relacionamento da tabela *Aluno* e *Curso*: uma junção para *Aluno* e *Matriculas* e outra para *Matriculas* e *Curso*. Caso não encontre um caminho no grafo G ou exceda a quantidade de junções, o caminho será considerado inválido. O número da coluna na tabela de entrada indica o valor das chaves; as chaves 1 e 2 do primeiro caminho de mapeamento gerado, indicam os elementos extraídos de *L(1)* e *L(2)*. A mesma regra se aplica nas chaves 1 e 3, extraídas de *L(1)* e *L(3)* na Figura 4.2 (b); (as setas nas etapas (c), (d) e (e) da Figura 4.2 indicam a relação das chaves com os atributos).

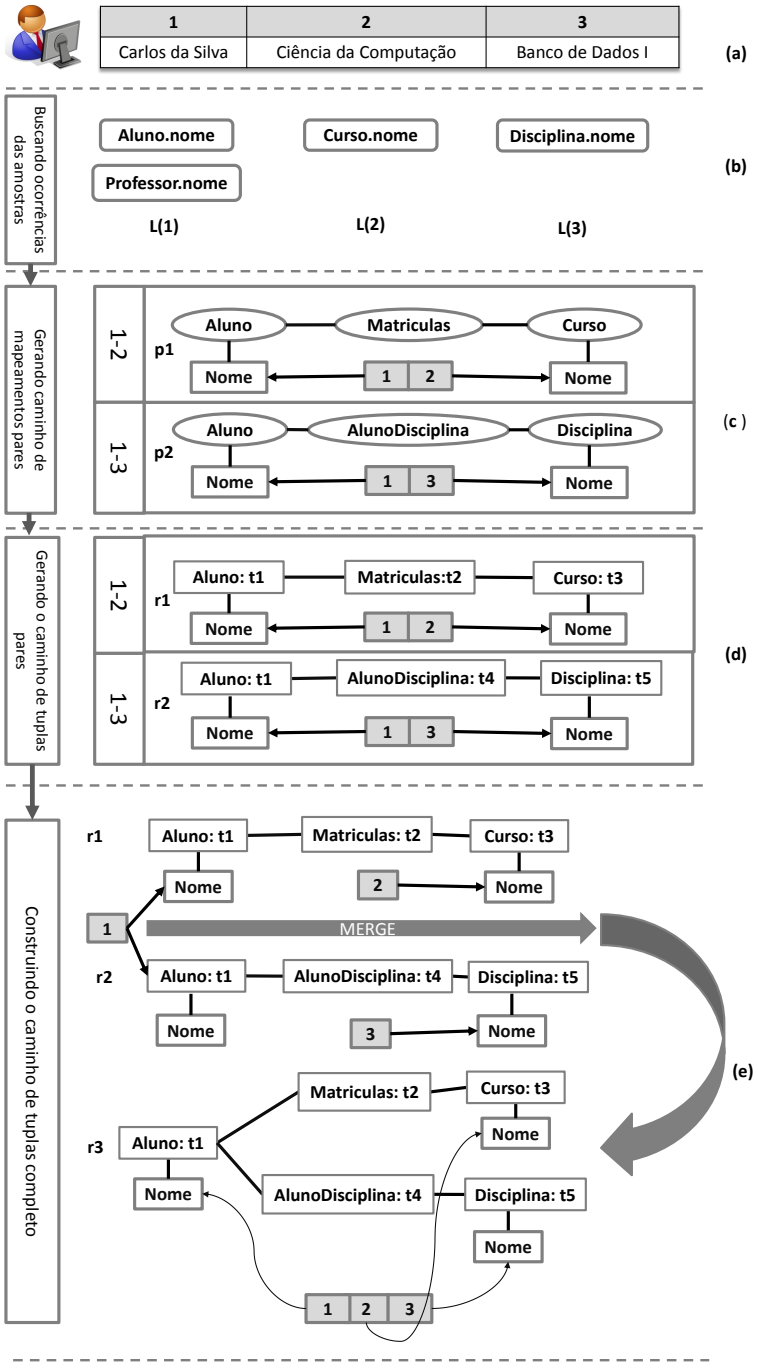

Figura 4.2: Exemplo do algoritmo *TPW*.

4.1.3 Geração do caminho de tuplas pares

Após gerar todos os caminhos de mapeamentos pares, é gerado o caminho de tuplas pares. Na linha 8 do Algoritmo 1, são adicionados a um caminho de tupla todos os caminhos

de mapeamento que sejam válidos. Um caminho de tupla verifica se um caminho de mapeamento é válido ou não. Para isso, é gerada uma consulta *SQL* (linha 7) para cada caminho de mapeamento. Se o retorno não é vazio, o caminho de mapeamento é válido, senão é inválido. Qualquer outro mapeamento par, que contém esse caminho, também é considerado inválido (primeira condição na linha 7). Caso contrário, é construído o caminho de tupla, para cada tupla retornada da consulta. Por exemplo, na Figura 4.2 (d) se no banco de dados de origem, a tupla *t1* contém *Carlos da Silva*, a tupla *t3* contém *Ciência da Computação* e *t2* liga *t1* e *t3* por chave estrangeira, essas tuplas são válidas, suportando o caminho de mapeamento *p1*; assim é gerado o caminho de tupla *r1*, que associa-se com suas chaves (1,2); da mesma forma *r2* é gerado. Assim, cada caminho de mapeamento válido resulta em um caminho de tupla.

4.1.4 Construção do caminho de tuplas completo

Após ter gerado os caminhos de tuplas pares, em seguida, é realizado a fusão entre esses caminhos (linhas 11 a 16 do Algoritmo 1). A partir de um mapeamento candidato M inicial (linha 11) é realizada a fusão com os caminhos de tuplas restantes. A fusão termina quando não é mais possível fundir dois vértices, retornando um valor negativo (linha 14).

Por exemplo, para que *r1* e *r2* (Figura 4.2 (d)) façam a fusão, é testado se o primeiro vértice de *r1* e de *r2* contém a mesma chave (o valor 1). Em caso afirmativo é verificado, então, se essas tuplas são idênticas. Caso forem idênticas, os vértices serão fundidos. Caso contrário, a fusão aborta e é criado um caminho de tupla base. Um caminho de tupla base é o resultado de uma fusão. Caso exista mais caminhos de tuplas pares, tenta-se a fusão com o caminho de tupla base, gerando um novo caminho de tupla base, até que não exista mais caminhos de tuplas pares para fundir. Por fim, além do mapeamento $r3$ da Figura 4.2 (e), outro mapeamento candidato seria gerado apenas substituindo a tabela $Professor$ pela $Aluno$.

O Algoritmo 1 faz a busca na base de dados por texto completo. O resultado é exibido em um formato parecido com o resultado final do algoritmo. Além disso, uma pontuação é mostrada a cada mapeamento candidato, formando um *ranking* entre os mapeamentos candidatos (linha 17), sendo a média de duas notas: a primeira nota indica a qualidade das amostras, coincidindo com os dados reais das tuplas e, a segunda é uma pontuação de complexidade, sendo o número de associações na correspondência. A nota qualifica cada mapeamento em relação a sua corretude e sua complexicidade.

4.2 *DISCOVER*

DISCOVER [Hristidis and Papakonstantinou, 2002] é uma ferramenta que resulta em redes de consultas candidatas oriundas de um plano gerador. Essas redes são criadas por um conjunto de tuplas fornecidas pelo usuário, estando associadas pela relação de chave-primária e estrangeira. *DISCOVER* realiza duas etapas para gerar as redes de consultas candidatas: (i) o plano gerador de redes candidatas gera todas as redes de consultas candidatas e, (ii) faz uma avaliação sobre as redes candidatas, a fim de reutilizar expressões em comum entre elas.

```
Entrada : Grafo de esquema G, Tamanho T, palavras-chave K;
Saída : Rede de consultas candidatas QC_s;
conjuntoTuplas CT_s;
k=selecionaChave(K);
foreach C in CT_s do
  | QC_s=geraRedesCandidatas(C,k);
end
foreach C in QC_s do
  | if C succeeds pruning then
  |   | QC_s ← QC_s - C;
  |   | else
  |   |   | continue;
  |   | end
  | end
end
return (QC_s)
```

Algorithm 2: *DISCOVER*

O Algoritmo 2 apresenta a implementação da construção das redes de consultas candidatas. Tem como entrada um grafo orientado G, representando o esquema da base de dados da Figura 4.1, um tamanho T, que limita o número de junções em uma rede, e o conjunto de palavras-chave K fornecidas pelo usuário. Já como saída, é apresentado o conjunto de redes candidatas que contém todas as palavras-chave sem redundância. Na linha 3, a estrutura CT_s contém todas as tuplas do banco que dados que continham as palavras-chaves. Na linha seguinte, é selecionada uma palavra-chave inicial de forma aleatória pela função *selecionaChave* que tem como parâmetro todas as palavras-chave. Após selecionada a palavra-chave, são geradas as redes de consultas candidatas (linha 6), através de uma busca no grafo G das palavras-chave e, armazenado o conjunto de redes candidatas na variável QC_s, realizada pela função *geraRedesCandidatas* que tem como parâmetro uma tupla C e o conjunto de palavras-chave K.

Após, são avaliadas as condições de eliminações dessas redes (linhas 9-14). Primeiro são eliminadas as redes candidatas que possuam relações repetidas com a mesma palavra-chave e, depois as redes em que suas folhas sejam relações que não possuam palavras-chave. Dados G

e K (conjunto de tuplas da Tabela 4.1), a construção das redes é feita em duas etapas principais: a geração do conjunto de todas as redes candidatas e a avaliação das redes candidatas.

	A	B	C
1	Carlos	Computação	
2	Jacson		Custos

Tabela 4.1: Amostras

A geração do conjunto de redes candidatas inicia lendo a primeira tupla de K, ou seja, *Carlos* e *Computação*. Assim, uma busca no grafo G é feita para encontrar um caminho entre essas palavras-chave. Para que o caminho ou a rede sejam válidos, eles devem conter todas as palavras-chave, a menos que existam tabelas repetidas como exemplo o caminho: $Aluno_{<Carlos>} \bowtie Matricula \bowtie Aluno_{<Jacson>}$. Um caminho como $Aluno_{<Carlos,Jacson>}$ $\bowtie Matricula \bowtie Curso_{<Computacao>}$ é considerado válido. São consideradas redes inválidas aquelas que não atenderem as condições de eliminação como: $Aluno_{<Carlos>} \bowtie Matricula \bowtie$ $Curso_{<Computacao>} \bowtie DisciplinaCurso$ (possui folha sem palavra-chave).

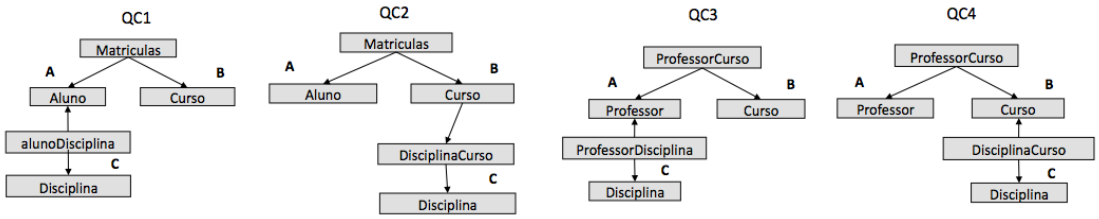

Figura 4.3: Consultas candidatas.

Na segunda etapa, *DISCOVER* avalia as redes candidatas para verificar quais são as redes inválidas e as redes candidatas que compartilham expressões de junções para construir um conjunto de expressões intermediárias e usá-las em outras redes candidatas. O plano gerador contém um plano de execução, que calcula e utiliza os resultados intermediários na avaliação das redes candidatas. Por fim, uma instrução *SQL* é produzida para cada tupla do plano de execução e estas instruções são passadas para o *SGBD*, retornando as redes que são as soluções para o problema. Para este exemplo, utilizando a tabela de entrada da Figura 4.1, o resultado são as redes candidatas *QC1*, *QC2*, *QC3* e *QC4* da Figura 4.3, pois atendem a todas as restrições.

Por fim, é feito um cálculo para o custo do plano de execução, definido pela seguinte fórmula: $\frac{frequencia^A}{log^B(tamanho)}$. Sendo a variável *frequencia* o número de ocorrências de $joins$ em uma rede candidata e a variável $tamanho$ é o número estimado de palavras-chave na rede candidata e, A e B são constantes pré-definidas. O termo $frequencia^A$ indica a quantidade

máxima de resultados intermediários reutilizados e o termo $log^B(tamanho)$ indica o tamanho dos resultados intermediários que são calculados primeiro. A variável A está relacionada ao tamanho das redes candidatas e, B a capacidade de reutilização, tendo assumido os valores para $\{A,B\}$=$\{1,0\}$.

Uma extensão do *DISCOVER* [Hristidis et al., 2003] foi proposta pelos mesmos autores e apresenta uma nova forma de classificar os melhores candidatos, através de técnicas de recuperação de informação ao invés do número de junções. A abordagem de *DISCOVER* será utilizada na próxima seção, como entrada para a ferramenta *FILTER*, com algumas modificações.

4.3 *FILTER*

FILTER [Shen et al., 2014] é uma abordagem de correspondência entre esquemas, baseada em amostras, que tem como objetivo descobrir o mínimo de consultas geradas. A correspondência é realizada através de alguns exemplos fornecidos pelo usuário, que podem representar a saída de uma consulta. Por exemplo, o usuário pode ter o conhecimento de algumas informações presentes na saída de uma consulta, essas informações serão utilizadas para gerar, automaticamente, sua consulta, podendo retornar mais de uma consulta. Dessa forma, o usuário pode utilizar estas consultas para complementar as informações da saída, cujos dados não tinha todo o conhecimento.

O algoritmo de geração das consultas candidatas em *FILTER* é o mesmo proposto por [Hristidis and Papakonstantinou, 2002], com uma pequena adaptação que rotula cada nodo da consulta em uma estrutura de rede, como apresentado na Figura 4.3, através do nome da coluna pertencente à tabela de entrada, que contém a amostra. Por exemplo, na consulta candidata *QC1*, a relação *Aluno* é rotulada com a coluna *A*. Além de *FILTER* são apresentadas mais duas ferramentas: *SIMPLEPRUNE* e *VERIFYALL*, que serviram como base para eliminar as redes de consultas candidatas inválidas. Todos esse algoritmos usam as redes de consulta. Nas subseções abaixo serão descritas e exemplificadas cada uma dessas ferramentas.

4.3.1 *VERIFYALL*

Com a tabela de entrada, representada na Figura 4.1 com algumas amostras, uma maneira de avaliar se a consulta é válida será verificar individualmente, se o resultado da consulta candidata equivale a cada uma das linhas da tabela de entrada. Desta forma, a

verificação será feita para cada linha e analisado seus resultados com um modelo de consulta *SQL* como a seguinte:

SELECT $*$ TOP 1
FROM $< Tabela >$
WHERE $< predicado >$
CONTAINS ($< amostra >$)

Por exemplo, usando o modelo de consulta *SQL* anterior, pode-se verificar a consulta candidata *QC1*, na Figura 4.3, para a segunda linha da tabela de entrada, na Figura 4.1, com a seguinte instrução:

SELECT $*$ TOP 1
FROM *Matriculas, Aluno, Curso, alunoDisciplina, Disciplina*
WHERE *Matriculas.idAluno = Aluno.idAluno* AND
Matriculas.idCurso = Curso.idCursoAND*Aluno.idAluno = alunoDisciplina.idAluno*
AND *Disciplina.idDisciplina = alunoDisciplina.idDisciplina* AND
CONTAINS (*Aluno.nome*, '*Jacson*') AND CONTAINS(*Disciplina.nome*, '*Custos*');

Se o resultado da consulta não for vazio, a consulta candidata *QC1* na Figura 4.3 contém a segunda linha da tabela. Portanto, é válida para esta linha, caso seja vazia é inválida para esta linha. Para verificar somente uma linha da tabela de entrada, usa-se *TOP 1* na cláusula *SELECT*. Desta forma, reduz-se o custo da transferência de dados do banco. Esta consulta *SQL* é chamada de verificação *QC-Linha*. Uma consulta é executada para cada linha da tabela de entrada. Se uma consulta *SQL* for inválida para uma linha, a consulta candidata é eliminada e não se verifica mais para as próximas linhas.

A ordem de execução das consultas candidatas não afeta o número de verificações realizadas, porém o ordem de linhas afeta, uma vez que as consultas aplicadas primeiro em linhas com menos células vazias, têm mais chance de falhar [Shen et al., 2014].

Por exemplo, a Figura 4.3 mostra as redes de consultas candidatas para a tabela de entrada, na Figura 4.1. Com isso, *VERIFYALL* realiza duas verificações *QC-linha* para *QC1* que satisfaz as duas linhas da tabela de entrada. Já *QC3* realiza uma verificação, pois na primeira

linha falha, sendo que *Carlos* não está na tabela *Professores*. Assim, têm três verificações entre essas duas consultas candidatas, caso se iniciasse a verificação pela segunda linha da tabela de entrada, o número de verificações aumentaria para quatro. Desta forma, constata-se que a ordem das linhas que serão verificadas altera o número de execuções do algoritmo.

4.3.2 *SIMPLEPRUNE*

Geralmente, a maioria das consultas geradas para as amostras são inválidas, pois poucas consultas representam a mesma relação entre cada tupla de amostras. Para resolver esse problema, o algoritmo *SIMPLEPRUNE* utiliza-se da existência de uma dependência de sub-redes entre os resultados das verificações de linha QC_s. Assim, ajuda na eliminação das consultas candidatas inválidas sem executar uma verificação de linha proposta em *FILTER*. Com isso, torna a eliminação das consultas inválidas mais rápida que a verificações de linha proposta em *VERIFYALL*.

Por exemplo, considere a tabela de entrada na Figura 4.1. *QC3* falha para a primeira linha. Isto implica que *QC4* também vai falhar para esta linha. Primeiro, porque a rede de consulta candidata *QC3* é uma sub-rede de *QC4*. Segundo, os rótulos *A* e *B* nas consultas candidatas *QC3* e *QC4*, são mapeadas para a instância dos mesmos atributos *Professor.nome* e *Curso.nome*. Uma vez que *QC3* não contém as tuplas cujos valores em *Professor.nome* e *Curso.nome* são '*Jacson*' e '*Computação*' em sua saída. Essa dependência entre as redes de consultas é chamada de "dependência de falha".

Se *QC3* for avaliado para primeira linha, *QC4* poderá ser eliminado. *SIMPLEPRUNE* utiliza da dependência de falha para salvar as verificações QC_s em uma fila. Desta maneira, primeiro é utilizado a fila de QC_s para verificar a dependência de falha para as redes de consultas candidatas e, caso não exista dependência, será feita a verificação de linha. Redes maiores são mais propensas a ter uma dependência de falha, assim é primeiro avaliado as redes menores que irão resultar em mais eliminações.

Para [Shen et al., 2014], o gargalo da abordagem era descobrir como acelerar a verificação das consultas candidatas. Para resolver isso foi proposto um método via filtros. Filtro é uma substrutura de uma consulta candidata. Um candidato só é válido se todos os seus filtros são válidos e, se um único filtro é inválido, implica que o candidato é inválido. Diferentes candidatos podem compartilhar filtros e avaliar filtros é menos custoso do que avaliar o próprio candidato. Por outro lado, o número de filtros é normalmente muito maior do que o número de

candidatos; por isso, é necessário selecionar os filtros criteriosamente para serem avaliados.

```
Entrada : Filtros F_s e consultas candidatas QC_s;
Saída : consultas candidatas válidas Q_s;
QX_s ← QC_s, FX_s ← F_s;
while QX_s ≠ 0 do
    F_i=selecionaProxFiltro(FX_s);
    F_i=avaliaFiltro(F_i), FX_s ← FX_s - F_i;
    if F_i succeeds then
        F_v ← F_v + F_i ;
        Q_i=verificaConsulta(QX_s , F_v), QX_s ← QX_s - Q_i;
        Q_s ← Q_s + Q_i;
        else
            F_f ← F_f + F_i;
            Q_i=verificaConsulta(QX_s , F_f), QX_s ← QX_s - Q_i;
        end
    end
end
return (Q_s)
```

Algorithm 3: *FILTER*

O Algoritmo 3 avalia as consultas candidatas. Tendo como entrada os filtros F_s que são subestruturas de uma consulta, e as consultas candidatas QC_s, gerando como saída as consultas válidas Q_s. No Algoritmo 3 para cada uma das consultas candidatas, primeiramente é selecionado um filtro por um modelo estatístico (linha 5), sendo realizado pela função *selecionaProxFiltro* que tem como parâmetro o conjunto de filtros FX_s. Logo depois é feita a avaliação do filtro escolhido (linha 6) gerando uma consulta *SQL* para cada linha da tabela. Na mesma linha, o filtro selecionado é excluído da lista dos FX_s.

Após a avaliação, é verificado se o filtro é válido (linha 7), ou seja, verifica se o valor retornado são os mesmos contidos na tabela de entrada. Desta forma, o filtro é adicionado ao conjunto de filtros válidos F_v (linha 8) e, no passo seguinte, a função *verificaConsulta* retorna as consultas não avaliadas QX_s que atendam o filtro F_v (linha 9). Assim, a consulta avaliada é excluída da lista de consultas QX_s e adicionada a lista de consultas válidas Q_s. Se o filtro for inválido, será adicionado ao conjunto de filtro inválidos F_f e testado com cada uma das consultas candidatas não avaliadas (linha 13). Por fim, o algoritmo termina quando todas as consultas foram avaliadas e retorna todas as consultas válidas Q_s.

A Figura 4.4 mostra três filtros das consultas candidatas *QC3* e *QC4* da Figura 4.3. Na avaliação de um filtro, um filtro é válido se a linha específica da tabela de entrada é dada como sendo condizente com o resultado de uma consulta *SQL* que é gerada para cada filtro. Esse filtro pode ser verificado utilizando a seguinte sintaxe:

SELECT $*$ TOP 1 FROM $< Tabela >$ WHERE $< predicado >$ CONTAINS ($< amostra >$)

Usando o modelo de consulta *SQL*, pode-se verificar o filtro *J1* para primeira linha da tabela de entrada (Tabela 4.1). O resultado da consulta não é vazio, portanto o filtro *J1* é considerado válido.

Após definir os filtros é necessário saber como utilizá-los. Por exemplo, considere a tabela na Figura 4.4, através da base de dados na Figura 2.1 e utilizando as consultas candidatas *QC3* e *QC4* da Figura 4.3. A Figura 4.4 apresenta três filtros J_1, J_2 e J_3 destes candidatos, com as colunas da tabela de entrada projetadas nas relações sublinhadas das consultas candidatas. As colunas *A*, *B* e *C* da tabela do exemplo são mapeadas para as colunas dos filtros. Os pares (J_i, j) são utilizados para caracterizar um filtro para a sub-rede de junção J_i na linha jth do exemplo, na coluna projetada na tabela.

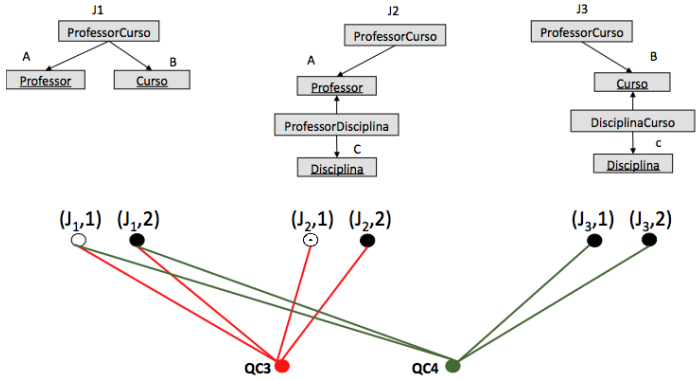

Figura 4.4: Utilizando filtros para verificação de candidatos.

Na Figura 4.4 são mostradas seis avaliações dos filtros, ou seja, o produto de três filtros com duas linhas da tabela de entrada da Tabela 4.1. Cada avaliação dos filtros $F = (J_i, j)$ está ligada a um candidato *QC*. Os círculos abaixo das avaliações dos filtros indicam filtro válido quando preenchidos, e quando não preenchidos inválido. O preenchimento não é conhecido antes de ser avaliado. Os dois candidatos *QC3* e *QC4* são todos inválidos. Para verificar isso é preciso apenas avaliar (J_1, 1). É avaliado se o filtro está contido na consulta candidata, sendo transformado em uma consulta *SQL* e,verificado se o seu resultado não corresponde com as entradas da tabela. Essa avaliação dos filtros com sucesso ou falha se trata de um problema combinatório. Por isso, segundo [Shen et al., 2014], o custo de avaliação é difícil de ser estimado de forma geral.

Para avaliar o custo, foi proposto um modelo probabilístico usado para saber qual o próximo filtro a ser avaliado, ou seja, o que tem mais probabilidade de falha. O custo é definido pela seguinte fórmula: $p(F) = \bar{p} \ . \ \frac{n_f}{|col(T)|}$, na qual $p(F)$ é a probabilidade da falha, $\bar{p}$ é uma constante que assume o valor da probabilidade média de falha, n_f é o número de células não

vazias em uma linha da tabela, que é mapeada para as colunas de um filtro, e $col(T)$ é o número total de colunas na tabela de entrada.

Como resumo do algoritmo de *FILTER*, através das redes de consultas candidatas são extraídos os filtros. Assim, esses filtros são avaliados com o objetivo de eliminar as consultas inválidas. Para ter uma ordem de avaliação dos filtros é utilizado um modelo probabilístico. Desta forma, consegue-se chegar as consultas válidas sem fazer às verificações de linha.

4.4 CONSIDERAÇÕES FINAIS

A Tabela 4.2 apresenta um comparativo das abordagens apresentadas neste capítulo para correspondência entre esquemas através de amostras. As seguintes características são consideradas na comparação: (i) a forma de classificar os resultados exibidos pelas ferramentas (coluna *Classificação*); (ii) a estrutura principal utilizada para execução dos algoritmos (coluna *Estrutura*); (iii) se utiliza como técnica de pesquisa no banco de dados aproximada ou por texto completo (coluna *Busca de ocorrências*); (iv) o número mínimo de entradas fornecidas pelo usuário para execução da ferramenta (coluna *Entradas mínimas*); e (v) quem gerencia a memória da ferramenta (coluna *Memória*).

Método	**Classificação**	**Estrutura**	**Busca de ocorrências**	**Entradas mínimas**	**Memória**
MWEAVER	Pontuação	Grafo dirigido	Texto completo	Prim. linha completa	SO
DISCOVER	N° Junções	Grafo dirigido	aproximado	Única amostra	SGBD
FILTER	Probabilístico	Grafo dirigido	aproximado	Única amostra	SGBD

Tabela 4.2: Comparativo das abordagens analisadas.

A forma de classificação utilizada para exibir os resultados para o usuário na ferramenta $MWEAVER$ é através de uma pontuação empregada a cada mapeamento candidato. Essa pontuação faz um $ranking$ de 0 até 1, apresentando os resultados em ordem decrescente, assim os resultados mais próximos de 1 são considerados mais corretos. Já na ferramenta $DISCOVER$ os resultados são classificados pelo número de junções, sendo que os resultados com o menor número de junções são exibidos primeiro para o usuário. $FILTER$ não classifica seus resultados, pelo fato dos filtros válidos e inválidos não serem conhecidos e, selecionados por um modelo estatístico afetando uma possível ordem das consultas candidatas.

Uma das características utilizadas pelos algoritmos para a redução do acesso ao banco de dados relacional e das operações de junções é a utilização da estrutura de um grafo dirigido. Neste grafo, as arestas indicam as relações de chave-estrangeira e os nodos as tabelas, pois

a busca no grafo que está em memória, torna-se muito mais rápida computacionalmente e, diminui a quantidade de acessos ao SGBDR.

Outra característica importante é a forma de armazenamento dos resultados intermediários obtidos pelos algoritmos. $MWEAVER$ delega a gestão de memória dos seus resultados intermediários para a memória principal do sistema operacional. Já $DISCOVER$ e $FILTER$ armazenam os seus resultados intermediários em tabelas temporárias no banco de dados, delegando assim a gestão de memória para o $SGDB$. Desta forma, $MWEAVER$ perde em desempenho devido uma grande quantidade de memória utilizada e a troca de dados entre o disco e a memória.

Por fim, as ferramentas se diferenciam quanto à forma de busca das ocorrências de amostras na base de dados. $MWEAVER$ usa uma busca pelo texto completo, o usuário precisa informar o valor exato que quer encontrar na base de dados. Entretanto, as outras duas abordagens relacionadas na Tabela 4.2 fazem a busca pelo texto aproximado, o usuário sabendo parte do valor já é o suficiente para fazer a busca.

Durante este capítulo foram apresentadas as características e as formas de trabalho de algumas abordagens de correspondência entre esquemas a nível de elemento. Cada uma das técnicas possui uma forma distinta de realizar o processo de mapeamento (embora existam algumas semelhanças). Contudo, em nenhuma das abordagens encontradas foi utilizado o dicionário de dados para gerar o grafo do esquema durante a execução do algoritmo.

Neste trabalho será utilizado o dicionário de dados de um banco de dados relacional e, também será aprimorado o espaço de busca, tanto no grafo quanto nos resultados intermediários. O algoritmo *TPW* e *VERIFYALL* serviram de referência para o restante de sua execução. Este trabalho visa explorar essa abordagem, de forma a verificar as diferenças desta técnica sobre as demais.

5 MÉTODO PROPOSTO

Este trabalho é baseado nas abordagens propostas em [Qian et al., 2012], [Shen et al., 2014] e [Hristidis and Papakonstantinou, 2002] e visa a criação de uma ferramenta que através de dados digitados pelo usuário (amostras) e uma base de dados relacional de origem cria automaticamente uma base de dados de destino e suas consultas correspondentes. Para tal, são realizadas 7 etapas: (i) conexão ao banco de dados e extração de um grafo através do dicionário de dados, (ii) obtenção da amostra informada pelo usuário, (iii) criação de critérios de seleção para diminuir o espaço de busca na base de dados, (iv) criação de possíveis esquemas baseados nas entradas, (v) definição das condições de eliminação dos mapeamentos inválidos, (vi) criação de consultas baseadas no esquema gerado e nas entradas, e (vii) classificação e exibição dos resultados em forma de visões.

Uma arquitetura do método proposto é apresentada na Figura 5.1. O Algoritmo 4 representa as etapas para geração dos mapeamentos candidatos. A entrada para o algoritmo é composta por dois parâmetros: uma base de dados D e um conjunto de amostras T fornecidas pelo usuário especialista para gerar um mapeamento. E como saída do algoritmo os mapeamentos e consultas no formato de visões.

Entrada : Base de dados D, amostras de usuário (T_1,...,T_n) ;
Saída : Mapeamento candidato MAP e consultas QRY;
Grafo G, TipoEntrada TE, respConsulta OCR, CaminhoPar $MPAR$, CaminhoCompleto MAP ;
$G \leftarrow$ GeraGrafo(D);
$TE \leftarrow$ ClassEntrada(T);
foreach i = *0;* $i > TE.length$ *;* $i++$ **do**
    $TC \leftarrow$ ClassTabela(TE_i, G);
    $OCR +=$ ConsultaAmostra(TC, TE_i);
end
$AMT \leftarrow OCR_1$;
foreach i = *1;* $i > OCR.length$ *;* $i++$ **do**
    if ! *CriaMapPar(*AMT, OCR_i, G, $MPAR$*)* **then**
        continue;
    end
end
$MAP \leftarrow MPAR_1$;
foreach i = *1;* $i > MPAR.length$ *;* $i++$ **do**
    if ! *weave(*MAP, $MPAR_i$*)* **then**
        continue;
    end
end
$QRY \leftarrow$ GeraConsula(MAP, OCR, G);
ranking(MAP, C);

Algorithm 4: *Mapeamento por amostra*

Conforme apresentado na Figura 5.1 e descrito anteriormente, o processo de mapea-

mento de esquema é formado por 7 etapas. Na primeira etapa (representada pelo número 1 na Figura 5.1) é feito o acesso ao dicionário do banco de dados D para extrair o esquema e construir o grafo dirigido G que representa o esquema de D. A função *GeraGrafo* (linha 4) do Algoritmo 4 gera o grafo G baseado no parâmetro de entrada D.

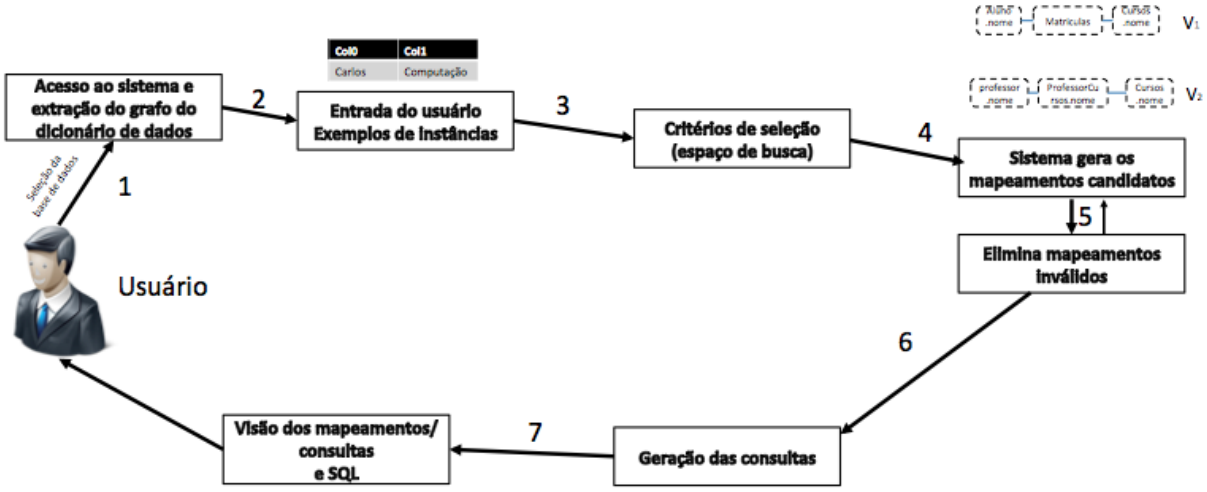

Figura 5.1: Arquitetura da proposta.

O grafo dirigido $G = (V, E)$ é definido como: V é um conjunto de nodos etiquetados com a tripla $(N, \{S(t,r)_1, \ldots, S(t,r)_n\}, nr)$, sendo N a etiqueta do nodo que corresponde ao nome de uma das tabela de D, $S(t,r)_i$ $(1 \leq i \leq n)$ é um o nome de um dos atributos de N, t identifica o tipo do atributo de S e r indica se o atributo é chave-estrangeira (*FK*) ou chave-primária (*PK*) ou nenhuma delas (*NULL*) e, nr é o número de tuplas de N. E é um par ordenado que corresponde a uma aresta que sai de um nodo V_1 apontando para um nodo V_2, sendo etiquetado pela tupla (fk, pk), onde fk é o nome do atributo correspondente a chave estrangeira de V_2 e pk é o nome do atributo correspondente a chave primária de V_1. A Figura 5.2 apresenta o grafo gerado a partir do banco de dados da Figura 4.1. Por exemplo, a tripla $(Aluno, \{idAluno(int, PK), nome(varchar, NULL)\}, 563)$ é a representação da tabela *Aluno* em um nodo de G. A aresta que faz a ligação com o nodo *Matriculas* contém a tupla $(idAluno(PK), idAluno(FK))$. Assim, G representa o esquema de D em memória. A construção de G é realizada através de consultas SQL ao dicionário de dados do *SGBD* que armazena D.

A segunda etapa (representada pelo número *2* na Figura 5.1) é a criação de uma tabela T a partir das amostras de entrada. T é formada por $T = (row, col)$, onde row é o número de linhas de T e, col é o número de colunas em T. Sendo que T inicia com $row = 1$ e col

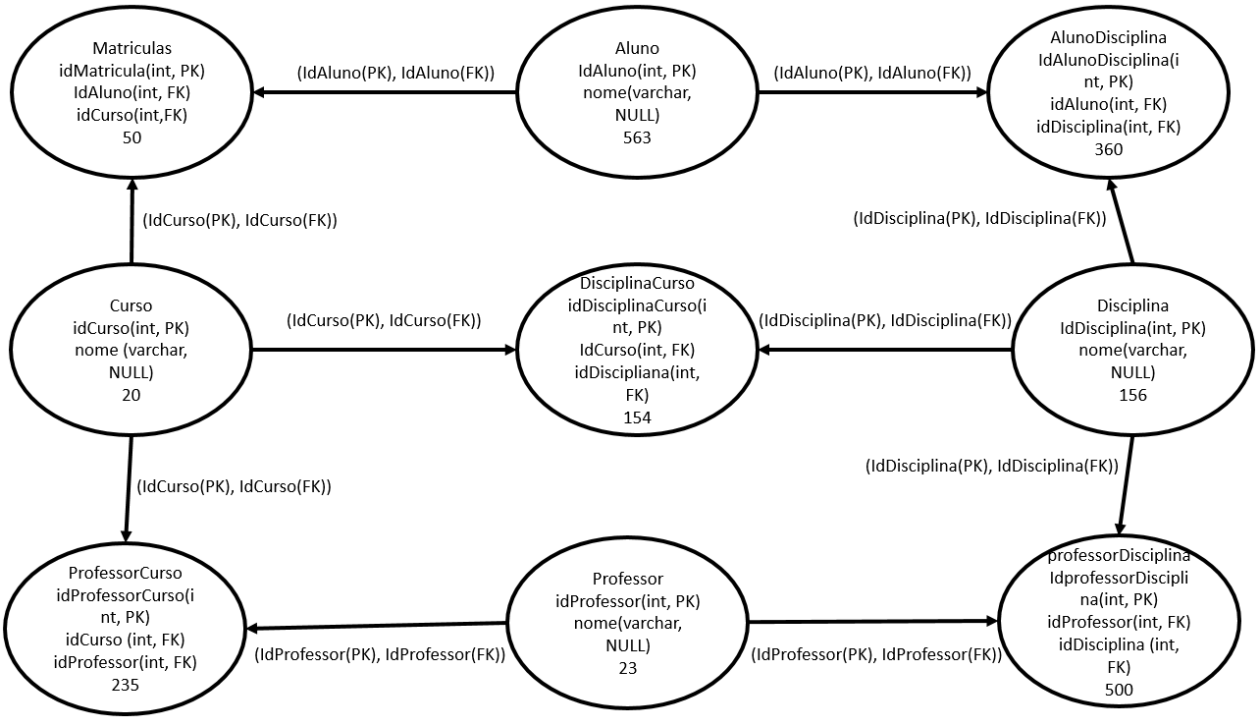

Figura 5.2: Grafo orientado G gerado a partir da Figura 4.1.

$= 2$, ou seja, tendo uma linha e duas colunas. As linhas e colunas podem ser adicionadas dinamicamente, possibilitando a inserção de novas amostras. As entradas em T possuem o domínio *S, I ou D*, onde S indica entradas do tipo *string*, I entradas do tipo *integer* e D entradas do tipo *date*. Essa classificação é realizada pela função *ClassEntrada* que tem como parâmetro as amostras T (linha 5) no Algoritmo 4, retornando a classificação em uma estrutura $TE = (amt, tpe)$, onde amt é a amostra e tpe o tipo da amostra. O tipo da entrada é identificado através de expressões regulares. Para o tipo *string* a expressão correspondente é $([A-Za-z0-9-]+)$, para o tipo *data* $((0[1-9]|[12]\d)\/(0[1-9]|1[0-2])|30\/(0[13-9]|1[0-2])|31\/(0[13578]|1[02]))$ e para o tipo *integer* $([0-9]+)$.

Após a construção de G e a classificação das amostras TE, é iniciada a terceira etapa (representada pelo número *3* na Figura 5.1). Nesta etapa, o objetivo é encontrar as tuplas em D que contenham as amostras de T. Assim, são identificados a tabela e atributo que contém a tupla na base de dados D, utilizando G para gerar uma ordem das tabelas que serão consultadas em D. O algoritmo proposto nesta etapa está dividido em duas partes: a primeira parte é a classificação das tabelas e atributos e a segunda parte é a geração das consultas SQL (linha 6 a 9, respectivamente, do Algoritmo 4).

A classificação das tabelas que serão consultadas é em ordem ascendente, seguindo os seguintes critérios: *(i)* a quantidade de chaves-estrangeiras QFK, *(ii)* a quantidade de atributos

QA do mesmo tipo da amostra em T e, *(iii)* a quantidade de registro QR.

O primeiro critério, QFK, tem o maior peso para classificação. O critério é construído da seguinte forma: percorrendo o grafo G é verificada a quantidade de atributos com chave-estrangeiras existente em cada nodo. A partir disso, as tabelas são classificadas do nodo que possui o menor número de chave-estrangeiras para o nodo que possui o maior número de chave-estrangeiras. Aplicando esta prioridade, as tabelas com o maior número de chave-estrangeiras serão as últimas a serem consultadas. Esse critério foi criado pois acredita-se que as tabelas com menos chaves estrangeiras são aquelas que armazenam dados mais relevantes da aplicação e, tabelas com muitas chaves estrangeiras têm o papel de fazer a junção entre as tabelas. Sendo assim, este conteúdo mais relevante vai estar nas tabelas com o menor número de chave-estrangeiras. Para as tabelas que possuam o mesmo número de chaves-estrangeiras é aplicado o segundo critério, reclassificando essas tabelas.

O segundo critério, QA, tem um peso menor que QFK para a classificação. Neste critério, é realizada uma busca em cada nodo do grafo G para verificar a quantidade de atributos que são do mesmo tipo da amostra em T. Desta forma, as tabelas são classificadas a partir daquela que contém o maior número de atributos para a que contém o menor número de atributos. Essas tabelas têm maior probabilidade de armazenar um dado correspondente a amostra. Para as tabelas que possuam o mesmo número de atributos é aplicada a terceira restrição, reclassificando essas tabelas.

O terceiro critério, QR, tem peso menor que o critério anterior para a classificação. Neste critério, é realizada uma busca em cada nodo de G e a classificação é através do número de registros nr de G. A classificação se dá tabela com maior número em nr para a que contém o menor número em nr. Se ainda houverem tabelas empatadas, o critério de escolha será aleatório.

Dado o grafo G representado pela Figura 5.2 e a amostra *"Jéssica"* como entrada, a classificação das tabelas aplicando os critérios ficaria na seguinte ordem:

1^o- Aluno

2^o- Disciplinas

3^o- Professor

4^o- Curso

5^o- ProfessorDisciplina

6^o- AlunoDisciplina

7^o- ProfessorCurso

8^o- DisciplinaCurso

9^o- Matriculas

As tabelas classificadas de 1-4, empatam no número de chaves-estrangeiras (que é zero) e no número de atributos do mesmo tipo (*varchar*) (que é um). O último critério faz com que a tabela *Aluno* seja a primeira da lista, pois tem o maior número de registros. Neste exemplo, na primeira tabela consultada *Aluno*, a amostra *"Jéssica"* seria encontrada no atributo *nome*. No Algoritmo 4, as tabelas classificadas são geradas pela função *ClassTabela* (linha 7) que recebe como parâmetro uma amostra classificada TE e o grafo G, armazenando o resultado em TC.

Na segunda parte da terceira etapa, é realizada a geração das consultas SQL (linha 8 do Algoritmo 4). A consulta das amostras é realizada pela função *ConsultaAmostra* (linha 8), que tem como parâmetro as tabelas classificadas em TC (linha 7) e a amostra a ser consultada TE_i. Cada consulta segue a seguinte cláusula *SQL*:

SELECT$< atributo >$
FROM$< tabela >$
WHERE$< atributo >$ LIKE $< \%amostra\% >$
LIMIT 1.

É usado o operador *LIKE* para que sejam buscadas partes da amostra, pois algumas entradas podem ser apenas uma parte da tupla que está em D e é usado *LIMIT 1* pois uma única tupla é o suficiente para identificar se a amostra existe no banco de dados ou não. Seguindo a ordem de classificação das tabelas, para cada atributo do mesmo tipo da amostra em cada tabela é gerado uma consulta *SQL*. O resultado de cada consulta que não retorna vazio é armazenado em uma estrutura $OCR = (tab, atr, tpl)$, onde tab é o nome da tabela, atr é o nome do atributo e tpl é a tupla que resultou da consulta. O modelo de consulta apresentado é aplicado para todos os tipos de entrada (*string*, *date* e *integer*). Por exemplo, para uma consulta com a amostra *Ciência* em T, a estrutura resultante é *(Curso, nome, Ciência da Computação)*. Lembrando que uma amostra pode ser encontrada em mais de uma tabela. Para todas as amostras contidas em

uma coluna da tabela de entrada, a consulta não deve retornar vazio para nenhuma das amostras na coluna. Nessa abordagem, foi utilizado como base o algoritmo do *VERIFYALL* descrito em [Shen et al., 2014]. *VERIFYALL* verifica que a mesma consulta deve ser válida para todas as linhas. No algoritmo proposto as consultas são feitas por coluna e a mesma consulta deve ser válida para todas as linhas da coluna que está sendo verificada. Por exemplo, tendo na primeira coluna da primeira linha a amostra *"Jacson"* e na primeira coluna da segunda linha a amostra *"Carlos"*. A consulta na tabela *Aluno* não retorna vazio para *"Jacson"* e *"Carlos"*. Já quando consultada a amostra *"Carlos"* na tabela *Professor* o retorno é vazio. Assim, torna-se válida somente a consulta na tabela *Aluno*. Para o exemplo com a amostra *Ciência* a cláusula *SQL* executada no banco de dados D será a seguinte:

SELECT $nome$

FROM $Curso$

WHERE $nome$ LIKE "$\%$*Ciência*$\%$"

LIMIT 1.

Com a terceira etapa finalizada, passa-se para a quarta etapa (representada pelo número *4* na Figura 5.1). Esta etapa possui como entrada o grafo G e a estrutura OCR gerada na etapa anterior e, a partir dessas entradas, cria os mapeamentos que são as visões da base de dados de destino. Com base no algoritmo TPW proposto em [Qian et al., 2012], mapeamentos intermediários são gerados em pares (linha 10 a 15 do Algoritmo 4), ou seja, as ocorrências contidas em OCR são correspondidas em pares. As amostras fornecidas na primeira coluna são usadas como base para combinar com as amostras das próximas colunas. Isso é realizado pois combinando todas as amostras com todas pode gerar caminhos pares duplicados e o número de combinações cresce exponencialmente. Assim, a ordem em que são inseridas as amostras pode gerar mapeamentos diferentes. Desta forma, uma nova estrutura do tipo $MPAR = (t_1, t_2, ..., t_n)$ é gerada para armazenar os resultados intermediários, sendo $MPAR$ um mapeamento par, onde $t_1, t_2, ..., t_n$ é o nome das tabelas que formam um caminho intermediário através de uma busca em G.

Por exemplo, em uma aplicação acadêmica que armazena seus dados na base de dados D e T contém duas amostras *"Carlos"* e *"Computação"*, dois resultados serão gerados para *OCR = (Aluno, nome, Carlos Silva) e (Curso, nome, Ciência da Computação)*, pois as tabelas *Aluno e Curso* contém as entradas de T no atributo *nome*. Com isso, é gerado um caminho intermediário *MPAR = (Aluno, Matriculas, Curso)*, pois através de uma busca no grafo G foi encontrado um

caminho entre *Aluno* e *Curso* por meio do nodo *Matriculas*. Após gerar todos os caminhos intermediários pares, é realizada uma fusão entre os caminhos pares. Para realizar a fusão são combinados os nodos que possuem os mesmos atributos de N. Assim, é criado o mapeamento de destino MAP, que é a fusão de todos os caminhos pares. Na Figura 5.3, é representada a geração de um MAP, sendo a partir das amostras de T *"Carlos", "Computação"e "Cálculo"* gerado os caminhos pares *MPAR(1) e MPAR(2)*. Com isso, é realizada a fusão entre os nodos iguais dos caminhos pares, desta forma gerando o mapeamento final MAP. No Algoritmo 4, a função chamada de *weave* (linha 18) realiza a fusão e tem dois parâmetros: O mapeamento par $MPAR$ e o mapeamento final MAP.

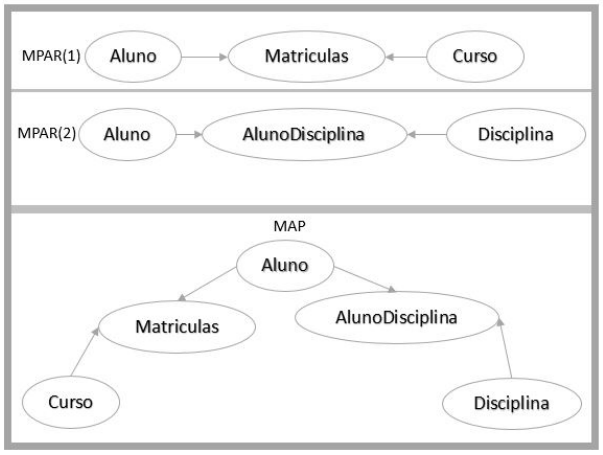

Figura 5.3: Geração de caminhos intermediários pares e caminho final.

Na quinta etapa (representada pelo número *5* na Figura 5.1) são realizadas algumas eliminações através de duas condições: a primeira verifica se existe um caminho em G para o par de ocorrência OCR. No Algoritmo 4, a função *CriaMapPar* (linha 12) tem como parâmetro uma ocorrência inicial AMT e uma segunda ocorrência OCR_i, o grafo G e $MPAR$. Caso exista caminho em G, ele é adicionado em $MPAR$, se não apenas passa para próxima verificação (linha 13). Se não existe caminho, o par OCR não é mapeado. A segunda condição elimina os mapeamentos pares gerados a partir de mais de um par de amostras que não são fusionados (condição da linha 18), ou seja, não possuem nenhum vértice com o nome igual a um vértice de outro mapeamento par.

Aplicando as condições de eliminação para gerar os possíveis mapeamentos, são geradas as consultas na sexta etapa (representada pelo número *6* na Figura 5.1). As entradas para está etapa são os mapeamentos MAP_s, as ocorrências OCR e o grafo G. Para cada mapeamento MAP é feita uma busca nas aresta de G para encontrar os atributos de chave-primária e chave-

estrangeira que fazem a ligação entre os nodos. Após, são selecionados as ocorrências tpl de OCR correspondentes aos nodos do mapeamento que foi realizado a busca. Com isso é gerada uma consulta SQL utilizando os atributos, tabelas e as ocorrências tpl, para gerar as consultas correspondentes na base de dados D. O resultado será a visão da consulta.

As visões das consultas são armazenadas na estrutura $QRY = (view_1, view_2, \ldots, view_n)$, sendo $view_i$ $(1 \leq i \leq n)$ a visão construída em ***SQL***, que é passada para última etapa (representada pelo número *7* na Figura 5.1). Dessas visões são exibidas algumas tuplas de exemplo para o usuário e um *link* onde terá acesso a visão da consulta em ***SQL***. No Algoritmo 4, a função *GeraConsulta* (linha 22) que tem como parâmetros os mapeamentos MAP, gera as consultas armazenando em QRY. As visões dos mapeamentos são representadas como V_1 e V_2 na Figura 5.1. A ordem de exibição dos resultados são os aqueles com o menor número de junções para o maior (linha 23), pois os resultados com menos junções são gerados inicialmente dada a menor complexidade. O processo é repetido até que o usuário pare de informar amostras.

Neste trabalho, não são tratados todos os possíveis tipos de dados que podem ser armazenado em uma base de dados, pois o maior parte dos dados nos bancos selecionados para este trabalhos são do tipo *string, date ou int* e são suficientes para gerar os mapeamentos. A utilização de mais de uma base de dados do mesmo domínio para mapear as amostras também não é abordada devido a complexidade de mapear os dados em mais de um banco de origem.

A ferramenta proposta é baseada no método proposto em [Qian et al., 2012] e no algoritmo do *VERIFYALL* descrito em [Shen et al., 2014]. O método proposto em [Qian et al., 2012] faz uma busca exaustiva na base de dados para encontra as amostras de usuário, essa busca é pelo texto completo, ou seja, o usuário precisa informar o valor exato que quer encontrar na base de dados, além do grafo que representa o esquema é montado manualmente antes da execução do algoritmo. No método proposto, o grafo é montado em tempo de execução e utiliza o dicionário de dados para extrair as informações do esquema. A consulta das amostras é feita na base de dados utilizando a classificação das tabelas pelas restrições de chave estrangeira, tipo do dado e número de registro, desta forma diminuindo o espaço de busca na base de dados. Também no método proposto, o valor da amostra pode ser parcialmente encontrado em uma tupla na base de dados, auxiliando o usuário que não tenha conhecimento total dos dados que estejam armazenados na base de dados. O algoritmo *VERIFYALL* [Shen et al., 2014] é aplicado na validação das consultas a partir das das amostras. No método proposto, são validadas as

amostra antes de gerar os mapeamentos, assim para as consultas geradas posteriormente aos mapeamentos não será necessária nenhuma verificação pois se foi gerado um mapeamento a partir das amostras validadas a relação entre as amostras existe, então é necessário apenas gerar as consultas a partir dos mapeamentos.

O método propõe 7 etapas para realizar o mapeamento entre as amostras e uma base de dados de origem. Inicialmente é selecionado uma base de dados e gerado o grafo que representa o esquema. Então a partir das amostras, classificação e eliminação são gerados os mapeamentos, que utilizam as tabelas classificadas e eliminando os mapeamentos inválidos. Por fim, as consultas são criadas e exibidas em formato de visão, juntamente com os mapeamentos correspondentes.

Neste capítulo foram apresentadas as características do método que ilustram seu funcionamento. O próximo capítulo trata da forma pela qual o método foi implementado.

6 IMPLEMENTAÇÃO

Neste capítulo é apresentada a maneira pela qual a ferramenta proposta, chamada *JMatching*, foi implementada. Como descrito no capítulo anterior, o método proposto neste trabalho é composto por 7 etapas representadas na Figura 5.1. Primeiramente, o *JMatching* vai receber como entrada a base de dados D, selecionada a partir de uma *interface web* pelo usuário especialista. A partir do esquema da base selecionada será criado o grafo dirigido G. Na segunda etapa o usuário insere as amostras em uma tabela de entrada. As amostras e G são a entrada para a terceira etapa, que classifica as tabelas de D e faz a busca das amostras em D. Com o resultado da busca na quarta etapa os mapeamentos são gerados e, na quinta etapa, ocorre a eliminação de mapeamentos inválidos. A partir dos mapeamentos são produzidas as consultas na sexta etapa. A última etapa é responsável por exibir as visões dos mapeamentos e das consultas. Abaixo será apresentada de forma breve a implementação das etapas do *JMatching*.

A interação do usuário com *JMatching* se dá meio de uma *interface web* gerada a partir do *framework Bootstrap*[1]. Foi utilizado o *Bootstrap* pois suas classes de estilo *CSS* auxiliam na criação de uma página com boa aparência para o usuário e garante que a página seja responsiva compatível com qualquer navegador. A *interface web* de *JMatching* é apresentada na Figura 6.1. Na primeira etapa o usuário especialista escolhe uma base de dados na opção *Select Database* (Figura 6.1 (F)). Com a base de dados selecionada, uma representação do esquema dessa base de dados é extraída através do dicionário de dados para o grafo dirigido G. O SGBD utilizado é o *Mysql* que armazena seu dicionário de dados em um banco chamado *information_schema*. Para extrair as informações necessárias para construção de G são consultadas as seguintes tabelas: *TABLES*, *COLUMNS*, *INNODB_SYS_FOREIGN* e *INNODB_SYS_FOREIGN_COLS* que armazenam, respectivamente, o nome das tabelas, as colunas de cada tabelas do esquema e os relacionamentos entre as tabelas. A partir dessa consulta são extraídos de *INNODB_SYS_FOREIGN_COLS* todas as colunas que fazem parte da relação. Assim, essas informações são mapeadas para os nodos e arestas de G. Por exemplo, o usuário seleciona a base de dados acadêmica D representada na Figura 4.1 e, através do dicionário de dados, o grafo dirigido G representado na Figura 5.2 é gerado. Será utilizado G e D para todos os exemplos desse capítulo.

Na segunda etapa através da *interface web*, o usuário insere as amostras em uma tabela

[1] http://getbootstrap.com/

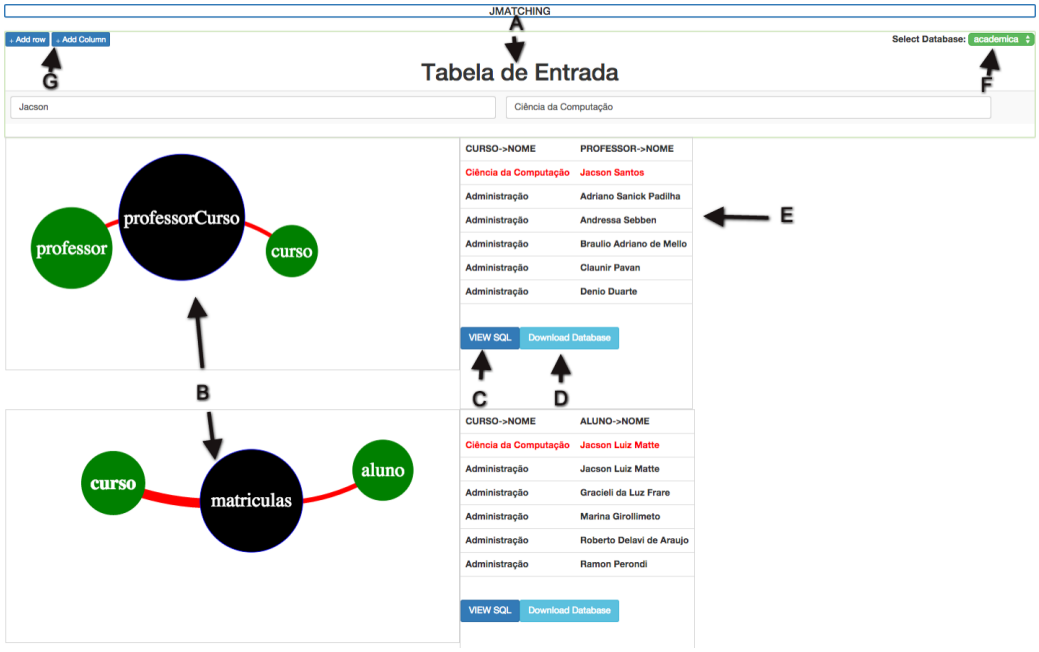

Figura 6.1: Resultado de um mapeamento no *JMatching*.

de entrada. Essa tabela inicial é composta por uma linha e duas colunas conforme representa a Figura 6.1 (A). O algoritmo utiliza pares de amostras então é necessário pelo menos duas amostras para execução do algoritmo. Caso o usuário queira inserir novas amostras, o botão *+Add Row* adiciona uma nova linha e *+Add Column* adiciona uma nova coluna (Figura 6.1 (G)). A coluna na tabela de entrada deve ser vista como o conteúdo de um mesmo atributo. Por exemplo, se o usuário inserir uma data, todas as linhas daquela coluna devem ser preenchida com datas. A leitura das amostras da tabela de entrada é realizada em *Javascript* e são enviadas via *AJAX*[2] para serem classificadas. As amostras são classificadas pelo seu tipo de atributo através de expressões regulares. Por exemplo, tendo como entrada do usuário *"Jacson"* na primeira coluna e *"Ciência da Computação"* na segunda coluna, essas amostras serão classificadas com o tipo *string*. Após inserir uma amostra, a tecla *Enter* deve ser pressionada para que esta reconhecida pelo *JMatching*.

A terceira etapa tem como entrada os resultados das etapas anteriores (amostra classificadas e o grafo G). A classificação e a busca são feitas por amostra. Como as duas amostras são do mesmo tipo (*string*), aplicando as restrições descritas no capítulo anterior a classificação das tabelas é a mesma para ambas as amostras. Para essa etapa a classificação segue a seguinte ordem: *(1-Aluno, 2-Disciplina, 3-Professor, 4-Curso, 5-ProfessorDisciplina, 6-AlunoDisciplina, 7-ProfessorCurso, 8-DisciplinaCurso e 9-Matriculas)*. Por exemplo, fazendo a busca para a

[2] http://api.jquery.com/jquery.ajax/

amostra *"Jacson"*, na primeira tabela *Aluno*, a mesma será encontrada no atributo *nome*. A amostra *"Jacson"* também será encontrada na terceira tabela *Professor* no atributo *nome*. E a amostra *"Ciência da Computação"* será encontrada na quarta tabela *Curso* no atributo *nome*. Assim, a estrutura resultante desta etapa seria *OCR = (*{*Aluno, nome, Jacson*},{*Professor, nome, Jacson*}, {*Curso, nome, Ciência da Computação*}*)*.

A quarta etapa possui como entrada o grafo G e a estrutura OCR. E a partir dessas entradas são criados os mapeamentos que utilizam como base a primeira amostra *"Jacson"*. Os resultados da primeira amostra *"Jacson"* serão combinados com os resultados da segunda amostra *"Ciência da Computação"*. Na Figura 6.2, o caminho intermediário $MPAR(1)$ é o resultado da busca em G do par *(*{*Aluno, nome, Jacson*} *e* {*Curso, nome, Ciência da Computação*}. E o caminho intermediário $MPAR(2)$ é o resultado do par *(*{*Professor, nome, Jacson*} *e* {*Curso, nome, Ciência da Computação*}. Após obter todos os caminhos pares, é realizada a fusão entre eles. Nesse exemplo a tabela de entrada possui apenas duas amostras, assim a combinação dessas amostras gera apenas caminhos pares em relação a um par de amostras (*Jacson* com *Ciência da Computação*). Com isso, os resultados obtidos por um único par de amostras são as representações de todos os mapeamentos produzidos por aquele par. Esses mapeamentos já são os resultados finais, pois não tem como realizar nenhuma fusão com outros mapeamentos pares gerados por outro par de amostras. Assim, resultados intermediários gerados a partir do mesmo par de amostra não são fusionados. Desta forma se faz necessário ter resultados intermediários gerados a partir de mais de um par de amostras para realizar uma possível fusão. Nesse exemplo, existe uma ambiguidade no resultado final, não é conhecido pelo algoritmo se *"Jacson"* é um aluno ou professor, resultando em $MAP(1)$ e $MAP(2)$ na Figura 6.2.

A quinta etapa não se aplica nenhuma condição de eliminação, pois este exemplo atende a todas as condições. Na sexta etapa são geradas as consultas. A entrada para está etapa são os mapeamentos MAP, as ocorrências OCR e o grafo G e o resultado é a visão das consultas. Seguido com os resultados dos exemplos anteriores, para gerar a consulta SQL do mapeamento, são necessárias algumas informações como: nome das tabelas e nome dos atributos. Para extrair essas informações são visitados diretamente os nodos de G que contém as tabelas de $MAP(1)$. O acesso é direto aos nodos de G, pois as tabelas envolvidas são conhecidas a partir de $MAP(1)$. Assim, extrai-se das arestas os nomes dos atributos que fazem a ligação entre as tabelas em $MAP(1)$ por chave estrangeira. Em OCR são extraídos os nomes dos atributos que serão usados na projeção da consulta e MAP contém o nome das tabelas.

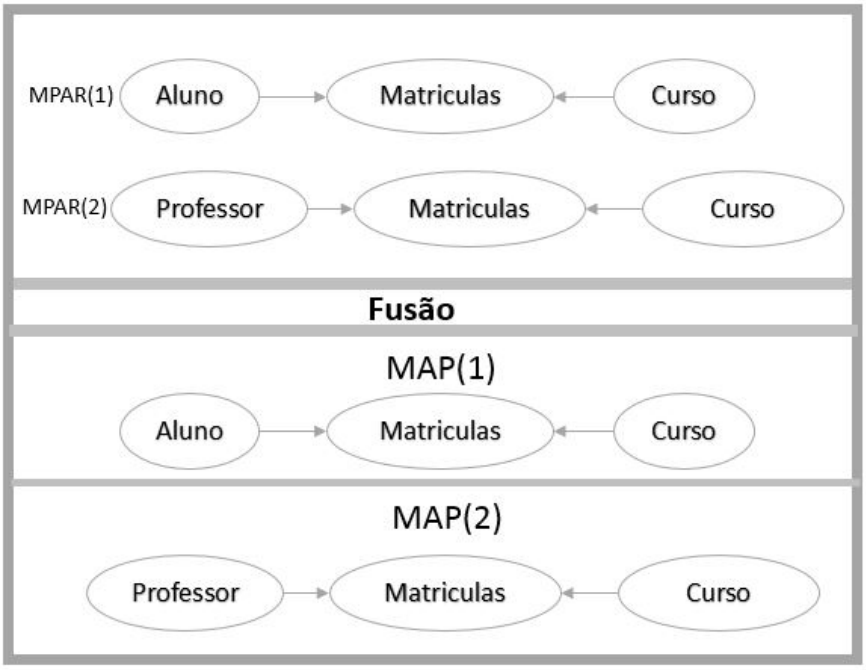

Figura 6.2: Geração do mapeamento final.

O mesmo procedimento de busca se aplica em $MAP(2)$. Com essas informações a seguinte visão da consulta é gerada para $MAP(1)$:

CREATE OR REPLACE VIEW $academico$
SELECT $Aluno.nome$, $Curso.nome$
FROM $Curso$, $Aluno$, $Matriculas$
WHERE $Aluno.id = Matriculas.idAluno$ AND $Curso.id = Matriculas.idCurso$

A partir das entradas da Figura 6.1 (A), as visões das consultas e os mapeamentos serão exibidos na sexta etapa como representado na Figura 6.1. Nesse exemplo, os dois mapeamentos possuem o mesmo número de junções, assim o primeiro resultado exibido é o que possui a tabela alunos e posteriormente o resultado que possui a tabela professores. Essa classificação é feita desta forma pois a primeira amostra encontrada foi na tabela alunos, gerando o primeiro mapeamento em relação a esta tabela. Para exibição dos mapeamentos (Figura 6.1 (B)) foi utilizada a biblioteca *Vis* [3] em *Javascript*. Essa visão dos mapeamentos é representada por um círculo com o nome da tabela e linhas que representam a ligação entre os círculos. Os círculos onde as amostras foram encontradas são preenchidos com a cor verde e o restante preenchido

[3] http://visjs.org/

com a cor preto. A biblioteca *Vis* utiliza *canvas* que é um elemento em *HTML5* que permite que se desenhem elementos gráficos usando *JavaScript*. Desta forma, o usuário pode interagir com os elementos contidos no *canvas* alterando a aproximação da visão do mapeamento e mudando a posição dos círculos. Essas interações são necessárias caso exista alguma dificuldade na visualização das visões dos mapeamentos.

Ao lado dos mapeamentos são exibidas algumas amostras de tuplas da base de dados (Figura 6.1 (E)). Essa visão representa o resultado da consulta em formato de tabela. Essa tabela contém na primeira linha o nome da tabela ao lado apontado por uma seta o nome do atributo e o restante das linhas preenchidas com algumas tuplas. O resultado da visão da consulta que esta em vermelho na Figura 6.1 (E) representa que as amostras fornecidas pelo usuário tinham relação. *JMatching* ainda tem mais duas opções para o usuário. Uma das opções é através do botão *VIEW* (Figura 6.1 (C)), onde pode ser visualizada e copiada a visão em *SQL* conforme representado na Figura 6.3. Essa visão do *SQL* contém uma clausula para criar ou sob-escrever alguma visão com o mesmo nome (*CREATE OR REPLACE*). Esse nome é criado concatenando o nome da base de dados com o número de classificação do resultado como *academica1* na Figura 6.3. O restante da visão da consulta contém o *SQL* com a projeção dos atributos que corresponde as amostras e faz a junção das tabelas que fazem parte do resultado através das suas relações. A outra opção é através do botão *Download Database* (Figura 6.1 (D)), onde o usuário pode baixar um arquivo da base de dados gerada no formato *CSV* informando o nome do arquivo e clicando em *gerar* conforme a Figura 6.4.

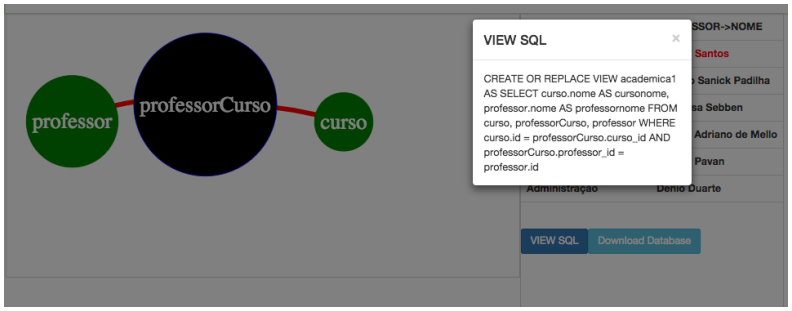

Figura 6.3: Visão em *SQL* do mapeamento em *JMatching*

Com esses dois resultados obtidos por *JMatching*, o usuário pode inserir mais uma linha através do botão *Add row* para tentar eliminar a ambiguidade ou mais uma coluna através do botão *Add Column* caso queira acrescentar mais alguma informação. Mesmo adicionando mais uma coluna a ambiguidade continua, pois a primeira amostra fornecida *Jacson* é encontrada como professor e aluno sendo ela a base para o mapeamento com as outras amostras. Levando

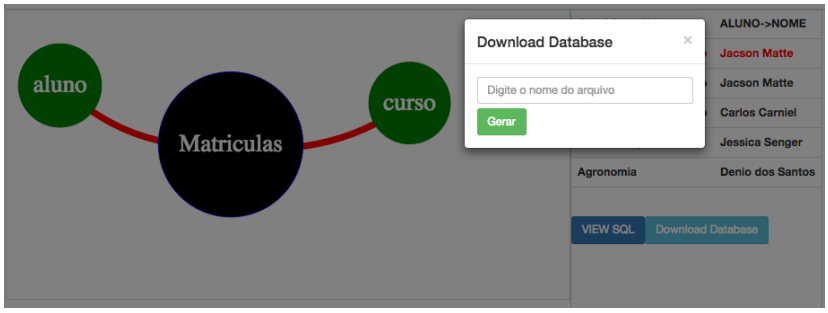

Figura 6.4: *Download* da base de dados gerado por *JMatching*

em conta que o usuário queira uma base de dados com os alunos e seus cursos, ele iria apenas inserir mais uma amostra na primeira coluna da segunda linha com o nome de um aluno. Por exemplo, suponha que a amostra "Carlos" é encontrada apenas na tabela *alunos*, isso geraria como resultado final apenas o mapeamento $MP(1)$.

Por fim o usuário especialista também tem a possibilidade de selecionar uma nova base de dado na opção *Select Database* (Figura 6.1 (F)). Ao selecionar a nova base de dados a tabela de entrada será reajustada para o estado inicial com duas colunas e uma linha e os dados de entrada anteriores serão excluídos.

Como descrito no método do capítulo anterior, *JMatching* tenta mapear através de técnicas de mapeamento por instâncias as entradas do usuário especialista para uma visão que as represente. Assim, essa representação se dá para uma base de dados de destino. Se tal base de dados de destino não é aceitável, o usuário especialista pode gerar novos mapeamentos inserindo mais amostras para que encontre um mapeamento aceitável. Para realizar o mapeamento, leva-se em conta de que o usuário especialista conhece somente as instâncias da base de dados e, caso ele tivesse que utilizar uma ferramenta de mapeamento em que é exigido o conhecimento da estrutura e semântica precisa dos esquemas, o usuário especialista enfrentaria dificuldades para criar um mapeamento. Mas para o usuário especialista a tarefa de apenas fornecer instâncias através de amostras pode tornar o mapeamento mais intuitivo e facilita a tarefa de integração de dados. Assim, para realizar o mapeamento o usuário especialista necessita apenas do conhecimento de algumas instâncias que estão na base de dados. Desta maneira, ao se chegar em uma solução aceitável para o problema de procurar amostra em uma base de dados, ela se torna eficiente o suficiente para atender aos requisitos interativos com o usuário, pois consegue realizar o mapeamento com um tempo de resposta aceitável. Isso se dá com o apoio de uma *interface web* que representa a visão de forma simples. Tanto a *interface* quanto as técnicas de mapeamento por instância têm o intuito de auxiliar os usuários especialistas a re-

alizar um mapeamento, obtendo as visões de várias formas. Assim, o usuário é isolado do nível conceitual, trabalhando apenas no nível externo da arquitetura três camadas do banco de dados. Desta forma, é possível realizar a tarefa de mapeamento em que somente o conhecimento de instâncias do banco é o suficiente para realizar um mapeamento do nível externo para o nível conceitual.

Neste capítulo, foi apresentada de maneira breve a implementação do *JMatching*. Para a implementação da *interface web* foi utilizado o *framework Bootstrap* e a biblioteca *Vis* que criam a *interface* gráfica da ferramenta. Utilizando essas ferramentas, a *interface* fica mais flexível e usável para o usuário especialista, facilitando a entrada de dados e exibição dos resultados. Os dados de entrada são lidos pelo *Javascript* e enviados por *AJAX* para serem processados pela linguagem *PHP*[4]. Foi utilizado *PHP* pois facilita a manipulação de dados em estruturas de *arrays*, essas estruturas foram muito utilizadas para gerar o grafo e manipular os resultados intermediários. E também pela fácil configuração do *PHP* com o *SGBD MYSQL* e o servidor *Apache* através do ambiente de desenvolvimento *XAMPP*[5]. No próximo capítulo serão apresentados alguns experimentos realizados no *JMatching*.

[4] http://php.net/
[5] https://www.apachefriends.org/pt_br/index.html

7 EXPERIMENTOS

Neste capítulo são apresentados resultados obtidos com a ferramenta proposta. Foram selecionadas duas bases de dados para os estudos de caso: a base *Acadêmica*, que representa os dados acadêmicos fictícios de uma universidade; e a base *Airbase*,[6] que representa a qualidade do ar de vários países da europa. A descrição completa de cada base será apresentada nas próximas seções. Para os estudos de caso serão abordadas as seguintes situações: (i) no caso C_1, a tabela de entrada contém uma linha e duas colunas preenchidas, ou seja, duas amostras, (ii) no caso C_2 a tabela de entrada contém uma linha e quatro colunas preenchidas, ou seja, quatro amostras, (iii) no caso C_3 a tabela de entrada contém duas linhas e quatro colunas, mas somente será preenchida com seis amostras e (iv) no caso C_4 a tabela de entrada contém duas linhas e quatro colunas preenchidas, ou seja, oito amostras. Alguns fatores foram relevantes para a escolha dos estudos de caso. Para o caso C_1 o fator abordado é o requisito mínimo para entrada da ferramenta, ou seja, duas amostras de entrada. No caso C_2 são adicionadas mais duas amostra para se obter um mapeamento mais completo, evitando o caso mínimo. Nos casos C_3 e C_4 será adicionado mais uma linha pois é o suficiente para se gerar um único resultado final para ambas as bases de dados, que pode ser o esperado pelo usuário. A partir dessas entradas serão analisados os resultados obtidos, o tempo para geração dos resultados e a quantidade de memória ocupada. Os estudos de caso serão aplicados separadamente nas bases de dados selecionadas. E também serão descritas algumas informações dos esquemas selecionados que são relevantes na analise dos experimentos.

7.1 Base de dados Acadêmica

A base de dados Acadêmica é representada no modelo lógico da Figura 7.1 e seu esquema contém dez (10) tabelas. A Tabela 7.1 apresenta o número de atributos, instâncias e chaves estrangeiras de cada tabela da base de dados *Acadêmica*. Os dados contidos na base de dados Acadêmica são fictícios, pois foram criados com o objetivo de testar todos os possíveis casos na ferramenta proposta. Cada tabela possui um papel distinto no esquema. A tabela *aluno* representa dados de alunos, a tabela *curso* representa dados de cursos, a tabela *disciplinas* representa dados de disciplinas, a tabela *domínio* representa dados de domínios das disciplinas e a tabela *professores* representa dados de professores. O restante das tabelas tem apenas o papel

[6] http://www.eea.europa.eu/data-and-maps/data/airbase-the-european-air-quality-database-8

de fazer as relações entre as demais tabelas do esquema, como representado no modelo lógico da Figura 7.1.

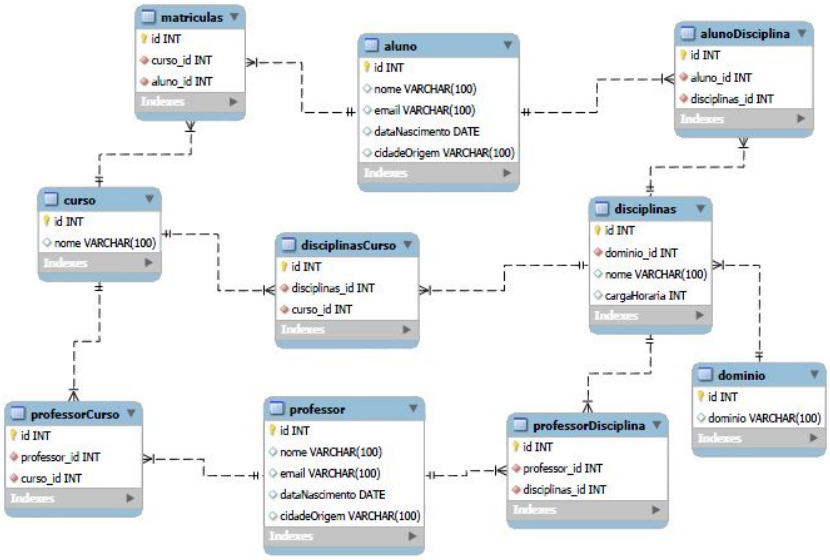

Figura 7.1: Modelo lógico da base de dados *Acadêmica*.

Para o caso C_1, suponha que o usuário queira a relação dos alunos e cursos presentes na base de dados Acadêmica. Assim, seria informado um nome de aluno e um nome de curso. Por exemplo, são inseridas as tuplas *Jacson* referente a um aluno conhecido pelo usuário e *Ciência da Computação* para um curso. Os resultados obtidos são mostrados na Figura 7.2. Os dois resultados não são esperados pelo usuário, um deles é a visão resultante com dados de professores e o outro é a visão do *email* em ambos os resultados.

Tabelas	Nº de atributos	Nº de instâncias	Nº de Foreign Key
aluno	5	20	0
curso	2	13	0
matriculas	3	56	2
disciplinas	4	47	1
alunoDisciplina	3	150	2
dominio	2	3	0
professor	5	13	0
professorCurso	3	78	2
disciplinasCurso	3	71	2
professorDisciplina	3	50	2

Tabela 7.1: Característica da base de dados *Acadêmica*.

Para o caso C_2, suponha que o usuário queira a relação dos alunos e suas cidades de origem, os cursos e as disciplinas presentes na base de dados Acadêmica. Assim, seria informado um nome de aluno, o nome de uma cidade, o nome de um curso e o nome de uma disciplina. Por exemplo, são inseridas as tuplas *Jacson* referente a um aluno, *Guatambú* referente a uma cidade, *Ciência da Computação* para um curso e *Algoritmos* para uma disciplina, todas conhe-

Figura 7.2: Resultado do experimento no caso C_1 na base de dados *Acadêmica*

cidas pelo usuário. Os resultados obtidos são representados pela Figura 7.3. Neste caso, alguns resultados também não são os esperados pelo usuário, mantendo o mesmos fatores errôneos de C_1. E as amostras de entrada não possuem relação por chave-estrangeira na base de dados para visão com professores, ou seja, não esta marcado a saída da consulta com a cor vermelha. Mas é gerada a visão com professores pois as tabelas que contém essas amostra possuem relação. E só é marcado em vermelho na visão da consulta se todas as amostras possuem relação por chave-estrangeira.

Para o caso C_3, suponha que o usuário ainda queira a relação dos alunos e suas cidades de origem, os cursos e as disciplinas presentes na base de dados Acadêmica. Mas para este caso será inserindo o nome de mais um aluno e um curso, para tentar gerar o resultado esperado. Por exemplo, são inseridas as tuplas *Jacson e Carlos* referente a dois alunos, *Guatambú* referente a uma cidade, *Ciência da Computação e Administração* para os cursos e *Algoritmos* para uma disciplina, todas conhecidas pelo usuário. Os resultados obtidos são representados pela Figura 7.4. Nota-se que não existe mais uma visão com professores, pois *Carlos* é um aluno, mas ainda tem como resposta não esperada pelo usuário o *email*.

Por fim, para o caso C_4, suponha que o usuário persista em obter os mesmo dados do caso anterior. Mas agora preenchendo toda a tabela de entrada. Neste caso são inseridas as tuplas *Jacson e Carlos* referente a dois alunos, *Guatambú e Chapecó* referente as cidades,

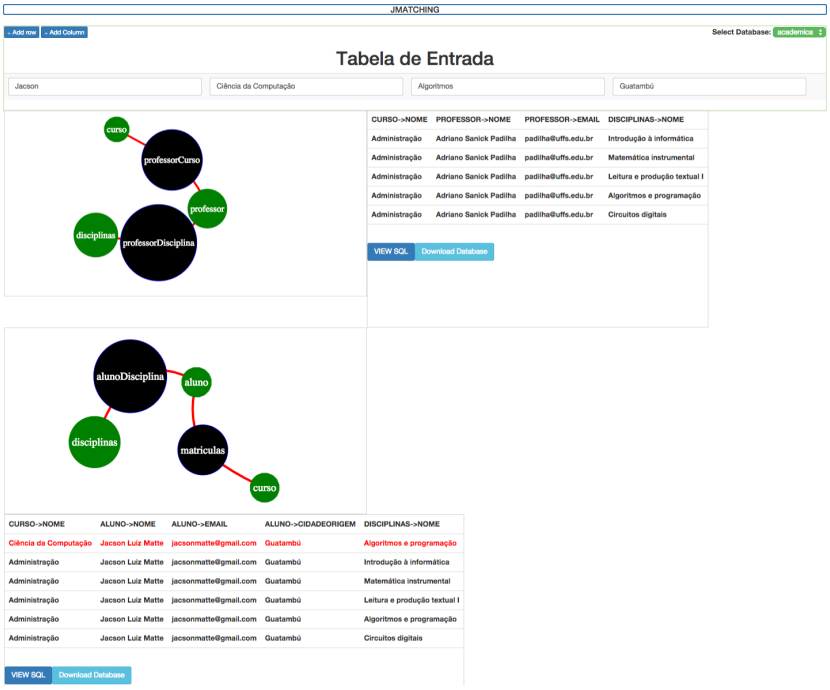

Figura 7.3: Resultado do experimento no caso C_2 na base de dados *Acadêmica*

Ciência da Computação e Administração para os cursos e *Algoritmos e Banco de Dados* para as disciplinas, todas conhecidas pelo usuário. Os resultados obtidos são representados pela Figura 7.5. Mesmo neste caso de teste ainda existe como resposta da visão o *email* que não é esperado pelo usuário.

7.2 Base de dados *Airbase*

A base de dados *Airbase* é representada no modelo lógico da Figura 7.6 e seu esquema contém doze (12) tabelas. Essa é uma base de dados real que armazena dados sobre a qualidade do ar na Europa, contendo dados de monitoramento da qualidade do ar e as informações dos países participantes de toda a Europa. A Tabela 7.2 apresenta o número de atributos, instâncias e chaves estrangeiras de cada tabela da base de dados *Airbase*. O propósito do conteúdo que cada tabela armazena são os seguintes: a tabela *measurement_configuration* representa os dados de configuração da medição do ar, a tabela *measurement_info* representa os dados de informações adicionais da configuração da medição do ar, a tabela *station* representa os dados das estações de medição do ar, a tabela *network_info* representa os dados que correspondem ao padrão de tempo de um grupo de estações, a tabela *station_info* representa as característica de uma estação, a tabela *meteorological_parameter* representa dados meteorológicos da estação, a tabela *country* representa os países, a tabela *airbase* representa os endereços eletrônicos que contém XML_s

Figura 7.4: Resultado do experimento no caso C_3 na base de dados *Acadêmica*

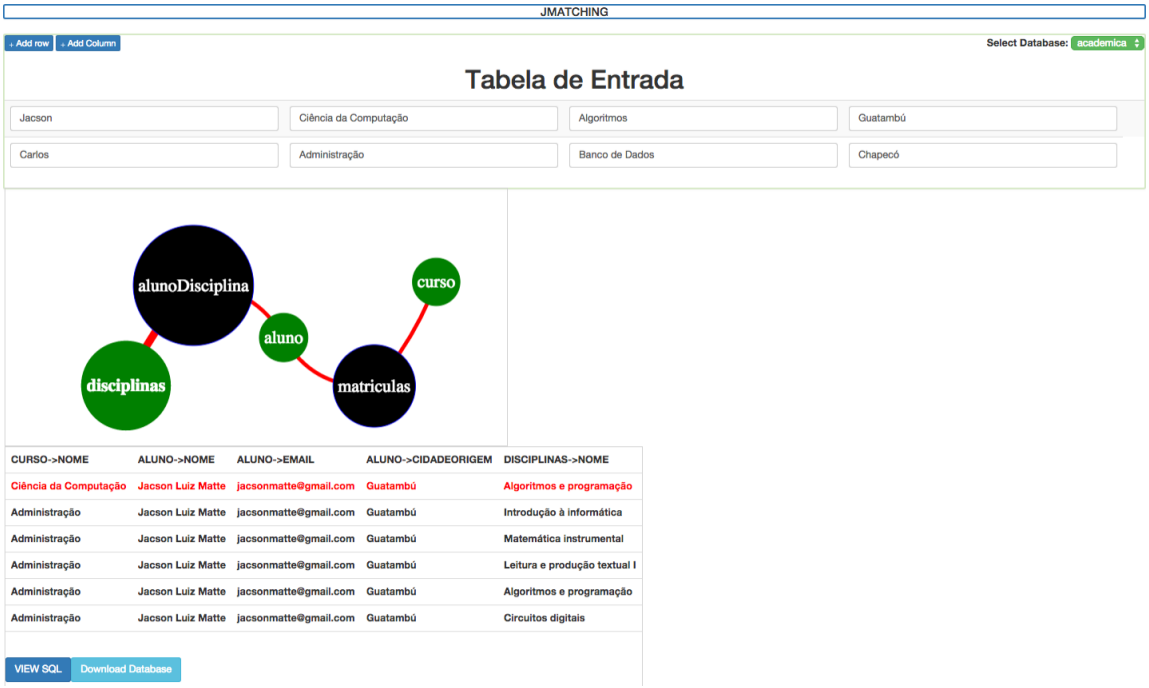

Figura 7.5: Resultado do experimento no caso C_4 na base de dados *Acadêmica*

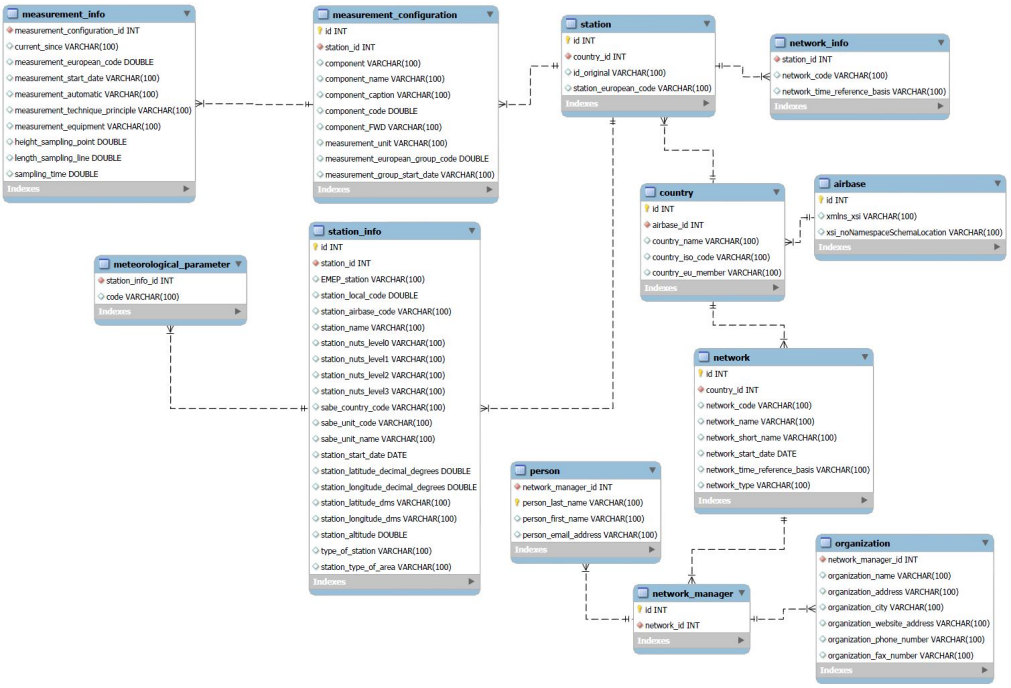

Figura 7.6: Modelo lógico da base de dados *Airbase*.

com informações dos países, a tabela *network* representa os nomes e características das redes entre regiões dos países, a tabela *person* representa os nomes e informações de quem gerencia uma *network*, a tabela *organization* representa as informações de localização das *networks* e, finalmente, a tabela *network_manager* representa a ligação entre as entidades *network, person e organization*.

Tabelas	Nº de atributos	Nº de instâncias	Nº de Foreign Key
measurement_configuration	10	11141	1
measurement_info	10	6807	1
station	4	2635	1
network_info	3	1702	1
station_info	21	1580	1
meteorological_parameter	2	1951	1
country	5	16	1
airbase	3	27	1
network	8	158	1
person	4	83	1
organization	7	97	1
network_manager	2	154	1

Tabela 7.2: Características da base de dados *Airbase*.

Para o caso C_1, suponha que o usuário queira a relação das características meteorológicas e as estações presentes na base de dados *Airbase*. Assim, seria informado um nome de uma característica meteorológica e um identificador de uma estação. Por exemplo, são inseridas as tuplas *Wind velocity* referente a um dado meteorológico conhecido pelo usuário e *Koliba* para o identificador de uma estação. Os resultados obtidos são representados pela Figura 7.7. Nota-se que além do identificador de uma estação retornou em ambos os resultados o nome da estação.

Figura 7.7: Resultado do experimento no caso C_1 na base de dados *Airbase*.

Para o caso C_2 suponha que o usuário queira a relação das características meteorológicas, as estações, os países e as redes presentes na base de dados *Airbase*. Assim, seria informado o nome de uma característica meteorológica, um identificador de uma estação, o nome de um país e o nome de uma rede. Por exemplo, são inseridas as tuplas *Wind velocity* referente a um dado meteorológico, *Koliba* para o identificador de uma estação, *Slovakia* para o nome de um país e *EMEP and Rural* para o nome de uma rede, todas conhecidas pelo usuário. Os resultados obtidos são representados pela Figura 7.8. Ainda existem dois resultados com o nome da estação que não são esperados pelo usuário, mas esse resultado pode ser interessante.

Para o caso C_3, suponha que o usuário ainda queira a relação das características meteorológicas, as estações, os países e as redes presentes na base de dados *Airbase*. Mas para este caso será inserido o nome de mais uma característica meteorológica e um identificador de uma estação, para tentar gerar o resultado esperado. Assim, são inseridas as tuplas *Wind velocity e Temperature* referentes a dois dados meteorológicos, *Koliba e Rudnany* para os identificadores de uma estação, *Slovakia* para o nome de um país e *EMEP and Rural* para o nome de uma rede, todas conhecidas pelo usuário. Os resultados obtidos são representados pela Figura 7.9. Neste caso a ferramenta proposta retorna o provável resultado esperado pelo usuário.

O resultado esperado foi resolvido no caso C_3, mas aplicando o caso C_4 o mapeamento pode ser melhor validado. Para este caso, serão inseridos o nome de mais de um país e o nome

Figura 7.8: Resultado do experimento no caso C_2 na base de dados *Airbase*.

Figura 7.9: Resultado do experimento no caso C_3 na base de dados *Airbase*.

de uma rede. Por exemplo, são inseridas as tuplas *Wind velocity e Temperature* referentes a dois dados meteorológicos, *Koliba e Rudnany* para os identificadores de uma estação, *Slovakia e*

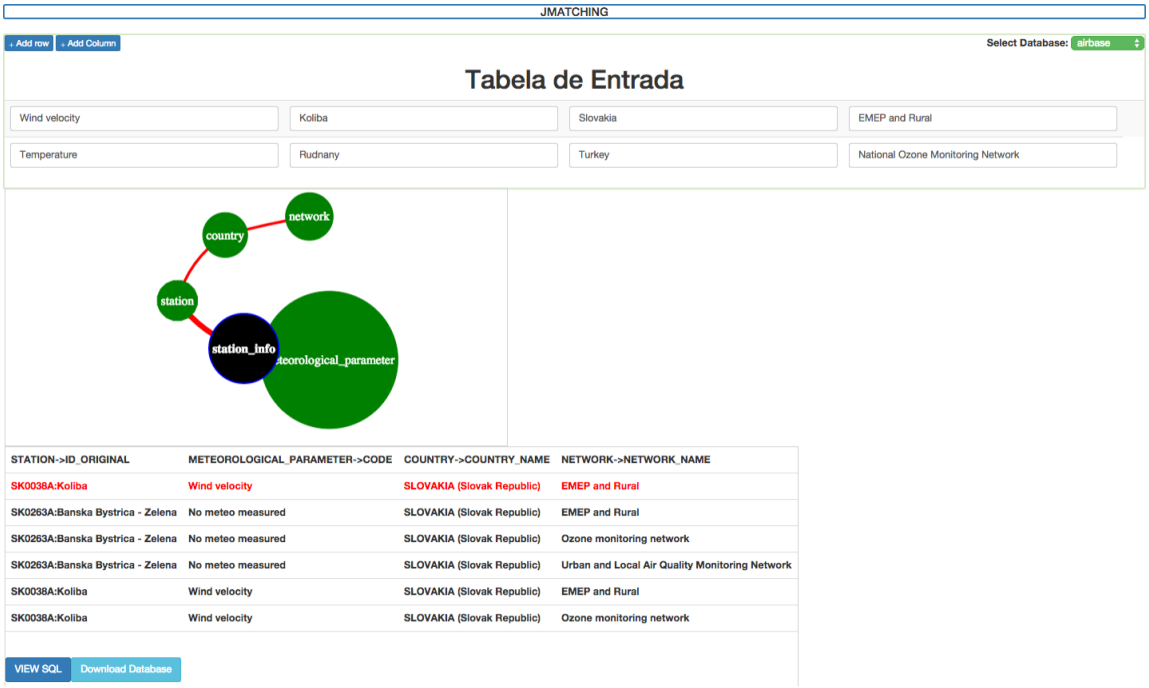

Figura 7.10: Resultado do experimento no caso C_4 na base de dados *Airbase*.

Turkey para os nomes de países e *EMEP and Rural e National Ozone Monitoring Network* para o nome de uma rede, todas conhecidas pelo usuário. Os resultados obtidos são representados pela Figura 7.9. Neste caso a visão retornada se mantém a mesma, pois os dados no caso anterior já eram o suficiente para gerar o mapeamento final.

7.3 Considerações finais

Baseado nos experimentos, pode-se concluir que com amostras mais genéricas os resultados esperados pelo usuário não são abrangentes. Por isso, quando a amostra for mais específica os resultados tendem a ser mais corretos. Conclui-se, também, que com algumas amostras consegue-se chegar a um único mapeamento como resposta. Caso o usuário não tenha conhecimento de amostras mais específicas, ele pode basear-se na visão da consulta para complementar o conteúdo das amostras de entradas. Assim, quanto mais específica for a amostra mais coerente será o resultado. Também percebe-se que mesmo que as amostras não possuam uma relação direta por chave-estrangeira na base de dados, a visão do mapeamento é gerada, pois as tabelas em que as amostras foram encontradas são relacionadas.

A Tabela 7.3 apresenta resultados extraídos a partir dos casos de teste aplicados com a base de dados *Acadêmica e Airbase*. Foram extraídos especificamente o tempo de execução e o consumo de memória da aplicação para gerar a visão com os casos de teste. Na base de

Acadêmico		
Casos	Tempo Execução (secs)	Consumo de Memória (Mb)
C_1	1,433788114	0.75
C_2	1,433789047	0.75
C_3	1,433789311	0.75
C_4	1,433873364	0.75
Airbase		
C_1	1,433889290	0.75
C_2	1,433947085	0.75
C_3	1,433947994	0.75
C_4	1,433958089	0.75

Tabela 7.3: Tabelas de testes de execução dos casos de uso.

dados *Acadêmica* percebe-se que o tempo de execução nos casos de teste não sofrem muita variação, mas conforme aumenta o número de amostras o tempo de resposta também aumenta. O mesmo acontece para base de dados *Airbase*, mas o primeiro caso possui um tempo maior em relação ao mesmo caso na base *Acadêmica*, pois a *Airbase* possui um número maior de tabelas, atributos e instâncias. A diferença é muito pequena não passando de 1 segundo para o tempo de execução entre os casos nas duas bases de dados. Assim, o tempo de reposta para o usuário não se torna muito significativo. Percebe-se também, analisando a Tabela 7.3, que o consumo de memória é o mesmo em todos os casos das duas bases de dados. Isso ocorres pois nas tabelas que foram encontradas as amostras são necessárias apenas uma instância por tabela, fazendo com que mesmo em bases maiores o consumo de memória seja o mesmo ou próximo de bases menores.

No decorrer deste capítulo foram apresentados alguns experimentos realizados com o *JMatching*. Inicialmente, foi aplicado os estudos de caso na base de dados *Acadêmica* e mostrado seus resultados. Posteriormente, foram aplicados os estudos de caso na base de dados *Airbase* e também mostrado seus resultados. Os resultados mostram que a proposta atenda os objetivos definidos para o trabalho: mapear esquemas coerentes com as amostras informadas pelos usuários especialistas. Por fim, foi apresentada a eficiência da ferramenta em relação ao tempo de execução e consumo de memória por meio de uma tabela.

8 CONCLUSÃO

A correspondência entre esquemas é um importante problema tratado por pesquisas em banco de dados. Sendo que é construída através de dois ou mais esquemas de origem que resultam em um esquema de destino. Geralmente, a correspondência entre esquemas é feita com auxílio de usuários conhecedores da aplicação e do banco de dados tanto no nível externo quanto no nível lógico. Porém, os usuários especialistas na aplicação mas sem conhecimento da estrutura do esquema do banco de dados, dificilmente conseguirão construir suas próprias correspondências. Neste trabalho [Matte, 2015], foram apresentadas algumas abordagens de correspondência entre esquemas a nível de elemento, a fim de auxiliar esses usuários especialistas [Qian et al., 2012, Shen et al., 2014, Hristidis and Papakonstantinou, 2002].

Com isso, baseado nestas abordagens, foi proposto um algoritmo para gerar mapeamentos candidatos a partir de amostras informadas pelo usuário especialista. A abordagem é constituída por 7 etapas, que compreendem respectivamente: a conexão ao banco de dados e extração de um grafo através do dicionário de dados; a obtenção da amostra informada pelo usuário; a criação de critérios de seleção para diminuir o espaço de busca na base de dados; a criação de possíveis esquemas baseados nas entradas; a definição das condições de eliminação dos mapeamentos inválidos; a criação de consultas baseadas no esquema gerado e nas entradas; e a classificação e exibição dos resultados em forma de visões. A implementação da abordagem proposta resultou na ferramenta *JMatching*. A ferramenta foi apresentada passo a passo utilizando um estudo de caso e foi utilizada em alguns experimentos de mapeamento de esquemas através de amostras.

Através dos experimentos percebeu-se que chega-se em uma solução aceitável para o problema de procurar amostra em uma base de dados. Também observa-se que a execução da ferramenta se dá em um tempo aceitável para o usuário especialista e independente do tamanho da base de dados, número de instâncias, tabelas e atributos. Assim, embora seja difícil para os usuários especialistas entender a semântica precisa de esquemas e mapeamentos, fornecer instâncias é muita mais intuitivo, sendo essa a motivação que levou a desenvolver a ferramenta proposta. Com ela, os usuários especialistas poderão estruturar seus próprios dados a partir de uma *interface web* com uma tabela de entrada que possibilita a inserção de amostras. A partir destas amostras ele pode criar ou modificar a visão do esquema de dados de uma forma que pode ser intuitiva, de modo que facilita integrar informações e criar seu próprio repositório

de dados, realizando a tarefa de correspondência entre esquema de forma automatizada. Isso isenta o usuário especialista da obrigação de compreender um esquema de origem complexo ou utilizar ferramentas de mapeamento que exijam essa compreensão.

A ferramenta proposta nem sempre chega a um mapeamento que seja esperado pelo usuário especialista, apesar de conseguir representar as entradas. Mas a corretude dessa representação tem como dependência principal as amostras do usuário especialista, pois quanto mais completa for a amostra em relação a instância na base de dados mais correta é a sua representação. Em casos em que as instâncias na base de dados possuam muito texto a ferramenta encontra dificuldades para gerar a representação do mapeamento esperada pelo usuário, pois essa busca na base de dados é realizada pelo operador *LIKE* da linguagem *SQL* que busca partes da instância. Assim, quanto menos representativa for a amostra menos representativa é a visão. A geração de caminhos intermediários são reduzidos, pois são gerados pela combinação da primeira coluna de amostras com o restante das colunas de amostras. Desta forma não se realiza a combinação de todas as amostras com todas, pois levaria a combinações ambíguas e cresceria exponencialmente o número de combinações dos resultados intermediários gerados a partir dessas amostras. Outro fato não abordado é a não utilização de mais de uma base de dados do mesmo domínio para realizar o mapeamento.

8.1 Trabalhos Futuros

Tendo como base a seção anterior, têm-se as seguintes perspectivas de continuação desse trabalho:

1º- Utilização de mais de uma base de dados do mesmo domínio;

2º- Adequação para que a busca na base de dados seja por texto completo ou partes do texto;

3º- Elaborar um aprimoramento do número de combinações dos resultados intermediários;

4º- Adaptar a ferramenta proposta para mais sistemas gerenciadores de banco de dados;

5º- Adaptar a ferramenta para reconhecer todos os domínios das amostras.

Essas perspectivas podem melhorar a ferramenta proposta, mas é importante ressaltar que atualmente a ferramenta realiza o mapeamento em tempo real e com pouca ocupação da memória.

REFERÊNCIAS

B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 133–144, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, Dec. 1986. ISSN 0360-0300.

P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *Proceedings of the 19th International Conference on Conceptual Modeling*, ER'00, pages 1–15, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3-540-41072-4.

P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. pages 130–145, 2003.

E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

R. Elmasri and S. Navathe. *Sistemas de banco de dados*. Pearson Addison Wesley, 2011. ISBN 9788579360855.

R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. 1997.

V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 670–681. VLDB Endowment, 2002.

V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proceedings of the 29th VLDB Endowment*, 2003.

A. L. JAMES and B. SHAMKANT. A theory of attribute equivalence in databases with application to schema integration.

P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 61–75, New York, NY, USA, 2005. ACM. ISBN 1-59593-062-0.

J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, volume 1, pages 49–58, 2001.

J. L. Matte. Correspondência entre esquemas orientada a amostras de tuplas. Technical report, Curso de Ciência da Computação. Universidade Federal da Fronteira Sul, Chapecó, 2015.

R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB*, volume 2000, pages 77–88, 2000.

L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 73–84, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9.

E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, Dec. 2001. ISSN 1066-8888.

R. Ramakrishnan and J. Gehrke. *Sistemas de gerenciamento de banco de dados - 3.ed.:*. McGraw Hill Brasil, 2008. ISBN 9788563308771.

Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 493–504, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2376-5.

P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. In S. Spaccapietra, editor, *Journal on Data Semantics IV*, volume 3730, pages 146–171. Springer Berlin Heidelberg, 2005a.

P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 4:146–171, 2005b.

A. Silberschatz, H. Korth, and S. Sudarshan. *Sistema de banco de dados*. CAMPUS - RJ, 2006. ISBN 9788535211078.

Y. Velegrakis, R. J. Miller, and J. Mylopoulos. Representing and querying data transformations. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 81–92, 2005.