UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VINÍCIUS CERATTI

Consultas Temporais em Banco de Dados de Grafos

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Profª. Dr. Renata Galante

Coorientador: MsC. Edimar Manica

Porto Alegre
2014

”Free your mind and find a way. The world is in your hands,
This is not the end.”

AGRADECIMENTOS

Às vezes, o universo parece trabalhar contra. São as pessoas presentes em nossas vidas que nos permitem passar por todas os obstáculos. A elas, eu dedico este trabalho. Primeiramente, agradeço aos meus orientadores, que motivaram a construção de um trabalho academicamente relevante, sobre um tópico recente (e novo para mim) que me permitiu não só a consolidação da minha graduação, como também me motivou a nunca parar de aprender, mesmo que o tempo pareça curto. Também agradeço aos demais professores da graduação e ao Instituto de Informática, pela reconhecida excelência do curso de Ciência da Computação. Agradeço aos meus chefes, pela paciência e compreensão, principalmente na etapa final deste trabalho. Agradeço também a todos os que estiveram presentes ao longo desses muitos anos de faculdade. Aos amigos que são quase família, e aos familiares que são bons amigos. Pessoas que, além de me motivar a continuar quando desistir parecia sensato, toleraram minhas ausências (as necessárias e as desnecessárias). E outras que, mesmo sempre distantes, se mostraram sempre atenciosas e estiveram sempre presentes. Aos pais e irmãos, agradeço pela presença incondicional e referência constante de porto seguro. Agradeço à minha madrinha, que me deu suporte constante mesmo à distância, à minha tia, com quem morei em parte desse tempo, e à minha irmã, que dividiu comigo os últimos anos e a história deste trabalho. Agradeço principalmente à minha mãe, que me ensinou os valores que eu levarei por toda a vida. Obrigado por me ensinar a tolerância, a paciência, a empatia, o respeito, a humildade, a educação e, acima de tudo, o amor incondicional.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
CSV	Comma Separated Values
ER	Entidade-Relacionamento
NoSQL	Conjunto de Bancos de Dados baseados em modelos não-relacionais
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
TSE	Tribunal Superior Eleitoral

RESUMO

Bancos de dados de grafos possibilitam a modelagem em uma estrutura orientada aos relacionamentos entre os dados e com poder de expressão e apresentação superiores aos tradicionais modelos relacionais. A evolução de SGBDs baseados nativamente em grafos já possibilita a aplicação de modelos complexos para a solução de problemas práticos, como o das consultas temporais. Assim, pode-se armazenar grandes conjuntos de dados com caraterísticas temporais, de maneira que consultas elaboradas possam obter informações expressivas sobre a história desses dados. O trabalho *"Time-varying social networks in a graph database: a Neo4j use case"* discute a construção de um modelo temporal para uma rede social construída a partir do registro de proximidade entre pessoas. Esse modelo pode ser abstraído e utilizado para a modelagem temporal de outros tipos de dados. Este trabalho se baseia nesse estudo para modelar as informações sobre os candidatos das últimas três eleições para vereador no Brasil. Apresenta-se a construção desta base de dados desde o início, o modelo de dados resultante e as consultas efetuadas sobre esta base, bem como os resultados dos experimentos realizados.

Palavras-chave: Bancos de dados de grafos, consultas temporais.

Temporal Queries in Graph Databases

ABSTRACT

Graph databases enable a relationship-oriented modeling structure, focusing on the relationships among data, providing a higher expression power than what is found on traditional relational models. The evolution of native Graph DBMSs allows the implementation of complex models to solve practical problems such as temporal queries. Therefore, large data sets with temporal features can be stored, in a way that queries could retrieve expressive information about the history of such data. This work is based on the paper "*Time-varying social networks in a graph database: a Neo4j use case*" that shows how to build a temporal model for a social network that represents the closeness events between people. In our work, we abstract the concepts in order to model temporal features and other data types. The main objective is to model information about the alderman candidates in the previous three elections in Brazil.

Keywords: graph databases, temporal queries.

1 INTRODUÇÃO

O presente trabalho apresenta uma análise sobre consultas temporais em bancos de dados de grafos. Grandes volumes de dados são armazenados popularmente em bancos de dados relacionais tradicionais que, apesar de serem padrão de mercado, ainda apresentam algumas deficiências no tratamento de informações com relacionamentos complexos. SGBDs baseados em grafos são uma alternativa para extensão do poder de expressão relacional, permitindo não só a visualização dos dados de maneira mais intuitiva como também a simplificação da escrita e leitura de consultas. Por seguirem os princípios do NoSQL, esses bancos de dados são plenamente adequados a informações da web e a métodos ágeis de desenvolvimento. Além disso, por serem relativamente recentes, motivam sua experimentação em trabalhos de pesquisa, procurando consolidar conceitos.

Bancos de dados de grafos também permitem a criação de modelos para consultas temporais, que são o objeto deste trabalho. Diversas aplicações apresentam a necessidade de consultas a informações históricas sobre o contexto que modelam, permitindo acessar uma configuração dos dados em um ponto específico no passado, observar a evolução ao longo do tempo ou até fazer projeções para o futuro, conforme os conceitos apresentados por Edelweiss (1998).

Diversos trabalhos consideram modelagens temporais usando grafos, por meio de várias alternativas: linguagens específicas para consultas temporais (RIZZOLO; VAISMAN, 2008) , adição da dimensão temporal nos rótulos das arestas de um grafo (GUTIERREZ; HURTADO; VAISMAN, 2007), ou armazenamento de *snapshots* dos dados associados a um instante no tempo (KHURANA; DESHPANDE, 2013). O trabalho de Cattuto et al. (2013) se destaca, entretanto, por desenvolver uma modelagem temporal usando um SGBD nativo de grafos. No modelo proposto, não há a necessidade de nenhum suporte temporal por parte do SGBD: a dimensão temporal é obtida com a inserção de um nodo que referencia um instante no tempo e tudo que estiver conectado a este nodo representa o estado dos dados nesse instante. Assim, consultas temporais são feitas com a seleção de um ou mais subgrafos, acessado por padrões de caminhamento no grafo principal.

Para tratar o problema da modelagem temporal, este trabalho consultou informações

a respeito das eleições para vereador no Brasil, considerando os últimos três pleitos (anos de 2004, 2008 e 2012). Essa escolha é relevante não só pela disponibilidade de dados validados por uma instituição oficial, mas também pelo volume de dados disponível e pelas caraterísticas temporais desejadas. Com esta base, é possível observar, por exemplo, a evolução das informações de um candidato que se candidatou em diversas eleições, ou as informações agregadas de um partido político ao longo dos anos.

O objetivo deste trabalho é, portanto, utilizar essas referências para modelar os dados propostos, ou seja, as informações sobre as eleições para vereador nos últimos anos. O resultado final deve ser, portanto, um banco de dados de grafos que permita consultas temporais diversas.

Para avaliar esse modelo, foram realizados experimentos diversos que abrangem vários tipos de consultas temporais possíveis. Através de uma metodologia específica, levantou-se informações sobre dos tempos de execução, mostrando como os dados temporais podem ser indexados com o modelo proposto e consultados de maneira eficiente. Além disso, com a demonstração do resultados de algumas dessas consultas, procurou-se elucidar a contribuição que o trabalho pode trazer.

Essa contribuição é importante pois permite e pode incentivar o eleitorado brasileiro a adquirir informações a respeito dos candidatos de seus municípios. Além disso, podem ser feitas análises de cunho social, pois os dados incluem informações relevantes nesse aspecto, como sexo e grau de instrução de candidatos. Partidos políticos podem, através desta base de dados, analisar o panorama eleitoral em que estão inseridos, bem como o seu desempenho a cada eleição.

Com este objetivo, parte-se de apresentação dos conceitos básicos sobre bancos de dados temporais, abordados pelo Capítulo 2. Com estes conceitos fixados, o Capítulo 3 discute suas aplicações a bancos de dados de grafos, através dos trabalhos relacionados, bem como motivações para o uso de bancos de dados de g rafos.

O Capítulo 4 apresenta, então, a construção do banco de dados temporal a partir dos dados disponibilizados pelo TSE para as eleições brasileiras, abordando todos os municípios do país. Neste ponto, discute-se problemas como ruídos encontrados e normalizações consideradas. Assim, após este pré-processamento, obtém-se o banco de dados pronto para experimentação.

No Capítulo 5, são relacionadas as diversas consultas avaliadas, explicando a semântica dessas consultas e apresentando visualmente o resultado das consideradas mais relevantes. Agrupam-se consultas com características em comum e discute-se o tempo de execução para elas e há comparativos entre diversas alternativas de construção. Apresenta-se um conjunto específico de consultas que mostra o valor agregado pelo modelo proposto ao permitir o levantamento de informações estatísticas abrangentes sobre o cenário eleitoral do Brasil.

2 CONCEITOS BÁSICOS SOBRE BANCOS DE DADOS TEMPORAIS

Modelos de bancos de dados tradicionais estão limitados à representação de um único estado dos dados - tipicamente, o atual. Este capítulo apresenta os conceitos básicos sobre bancos de dados temporais, que extrapolam esta limitação e dão maior poder de expressão à noção tradicional - e são objeto de estudo deste trabalho. Os próximos tópicos tratam da maneira como a temporalidade é representada, os aspectos importantes a considerar neste tipo de representação de dados e as diversas possibilidades de construções, bem como suas capacidades e limitações. Por fim, pondera-se os requisitos importantes para a realização de consultas sobre uma base temporal.

2.1 Representação Temporal

Em um banco de dados relacional tradicional, cada tabela é um conjunto de instâncias de dados, e cada coluna representa o valor corrente de um atributo de cada instância. Quando opera-se sobre esses dados, alterando um determinado atributo, perde-se o valor anterior e tem-se somente o último definido. Com a necessidade recorrente às mais variadas aplicações de preservar informações históricas, fez-se necessário desenvolver uma extensão ao modelo tradicional (EDELWEISS, 1998).

> Os modelos temporais acrescentam mais uma dimensão aos modelos tradicionais - a dimensão temporal. Esta dimensão associa alguma informação temporal a cada valor. Caso o valor de um atributo seja alterado, o valor anterior não é removido do banco de dados - o novo valor é acrescentado, associado a alguma informação que define, por exemplo, seu tempo inicial de validade. Todos os valores armazenados ficam no banco de dados. [...] Deste modo é possível acessar toda a história dos atributos, sendo possível analisar sua evolução temporal (EDELWEISS, 1998, p.227).

2.2 Aspectos de Temporalidade

Edelweiss (1998) discute diversos aspectos relacionados à representação temporal de dados. Nesta seção, destacam-se as propriedades mais usuais em modelos temporais.

Quanto à **ordem no tempo**, o mais comum é que se assuma que o tempo flui linearmente (*tempo totalmente ordenado*). Pode-se definir o tempo como *absoluto*, quando tem-se um tempo específico, como uma data, ou *relativo*, quando a validade está relacionada a outro ponto no tempo.

A **variação temporal** pode representar a natureza contínua do tempo, mas a representação *discreta* é a mais comum por simplificar a implementação. O tamanho dos intervalos definidos para essa representação discreta definirá sua **granularidade temporal**. Segundo Edelweiss (1998), são elementos primitivos da representação temporal:

- **instante no tempo** - representa um particular ponto no tempo. Em particular, tem-se o *instante atual (now)*, que define as noções de passado e futuro;
- **intervalo temporal** - o tempo decorrido entre dois instantes;
- **elemento temporal** - uma união finita de vários intervalos de tempo;
- **duração temporal** - fixa (como a duração de uma hora) ou variável (como a duração de um mês);
- **limites do tempo** - geralmente se referem a considerar ou não o tempo como infinito.

.

A representação temporal pode ainda ser classificada como *explícita*, através da associação da informação a um rótulo temporal, ou *implícita*, através da manipulação de conhecimentos sobre a ocorrência de eventos. O **tempo de transação** é o tempo no qual o fato é registrado no banco de dados, enquanto o **tempo de validade** se refere ao tempo em que o valor é válido na realidade modelada (EDELWEISS, 1998).

2.3 Tipos de Bancos de Dados Temporais

Edelweiss (1998) classifica os bancos de dados em quatro tipos diferentes: bancos de dados instantâneos, de tempo de transação, de tempo de validade e bitemporais.

Os bancos de dados **instantâneos** correspondem aos convencionais, onde são armazenados os valores presentes. A cada modificação no valor, o anterior é perdido. A manipulação temporal, neste caso, só pode ser realizada explicitamente, através dos programas de aplicação. Os bancos de dados de **tempo de transação** tratam informações temporais através da associação de rótulos temporais (*timestamps*) a cada valor. O estado atual do banco de dados é composto pelos últimos valores definidos, mas os valores

anteriores permanecem armazenados no banco de dados. Bancos de dados de **tempo de validade** associam a cada informação somente o tempo de sua validade no mundo real, sendo este obrigatoriamente fornecido pelo usuário. Não há acesso ao tempo em que uma informação foi definida, mas somente ao tempo em que é válida. Bancos de dados **bitemporais** fornecem a maneira mais completa de armazenar informações temporais, pois a cada informação são armazenados ambos os tempos de transação e de validade.

> Toda a história do banco de dados fica armazenada. É possível ter acesso a todos os estados passados do banco de dados - tanto a história das transações realizadas, como a história da validade dos dados. O estado atual do banco de dados é constituído pelos valores atualmente válidos. Valores futuros podem ser definidos através do tempo de validade, sendo possível recuperar o momento em que estes valores foram definidos para eventuais transações (EDELWEISS, 1998, p.236).

2.4 Consultas a Bancos de Dados Temporais

Para que se possa tirar proveito da dimensão temporal adicionada ao banco de dados, é necessário uma linguagem de consulta que permita recuperar todas as informações do banco, temporais e não temporais. Edelweiss (1998) define que essa linguagem deve ser enriquecida para manipular a dimensão temporal, permitindo fornecer valores de propriedades cujo domínio é temporal, se referir a determinados instante ou intervalo de tempo, recuperar valores com base em restrições temporais e fornecer informações temporais. O processamento dessas consultas pode apresentar problemas como o de indexação do grande volume de dados e à incompletude de informações (seja por incerteza quanto à existência de objetos em dado instante ou à indeterminação temporal).

Os bancos de dados instantâneos não permitem consultas temporais. Bancos de dados de tempo de transação permitem consultar valores atuais armazenados e valores definidos em tempos passados, sempre a partir do tempo de transação. Nos bancos de tempo de validade, é possível recuperar informações válidas em momentos presentes e passados, e também fazer estimativas para o futuro, analisando a história dos dados. Os bancos de dados bitemporais permitem que sejam feitas consultas sobre valores atuais, passados e futuros, considerando ambos os tempos de transação e validade. Como toda a história do banco é armazenada, pode-se obter, para um dado momento do passado, as informações de passado, presente e futuro válidas para este momento (EDELWEISS, 1998).

Consultas temporais possuem dois componentes ortogonais. O **componente de seleção** é representado geralmente por uma condição lógica (no caso da lógica temporal, usando operadores como *antes*, *depois* ou *início de intervalo*). De acordo com o tipo das condições estabelecidas, pode-se ter *consultas de seleção sobre dados*, *consultas de sele-*

ção temporal e *consultas de seleção mista*. O **componente de saída** consiste nos valores solicitados por uma consulta, sejam eles os dados ou informações temporais associadas aos dados. Pode-se ter, de acordo com o tipo das informações recuperadas, *consultas de saídas de dados*, *consultas de saídas temporais* e *consultas de saída mista* (EDELWEISS, 1998).

2.5 Implementação de Modelos de Dados Temporais

Um modelo de dados temporais deve contemplar todos os aspectos de uma modelagem clássica (relacional), sem nenhum comprometimento a eles, e acrescentar a possibilidade de representar informações temporais. Edelweiss (1998) considera importantes analisar, para um modelo de dados temporal, os seguintes aspectos:

- identificar o tipo de rótulo temporal utilizado pelo modelo (ponto no tempo, intervalo temporal, elemento temporal, duração);
- analisar a forma de variação temporal dos atributos (podem ou não variar com o tempo, todos ou alguns);
- verificar se os rótulos temporais são explícitos ou implícitos;
- homogeneidade temporal;
- apresentação e funcionalidades da linguagem de consulta;

A partir disso, enumera-se as seguintes informações a serem adicionadas ao modelo ER: tempo de início de um relacionamento, variação do relacionamento com o tempo, término do relacionamento, reincarnação de relacionamentos e restrições de integridade referencial com respeito à dimensão temporal. A dimensão temporal deve estar "embutida" no modelo. Deve-se permitir distinguir uma entidade temporizada de uma não-temporizada, bem como a associação entre estes tipos distintos; este relacionamento também pode ser definido como temporizado ou não temporizado. As restrições de cardinalidade devem considerar os pontos no tempo, mas se o relacionamento considerado for não temporizado, se mantem a semântica convencional.

2.6 Considerações Finais

Este capítulo apresentou as definições básicas para o desenvolvimento de um banco de dados temporal. Discutiu-se vários aspectos de temporalidade e da representação temporal, e foram apresentados os bancos de dados bitemporais (entre outros, é o que tem o maior poder de expressão) e os componentes de uma consulta temporal.

Uma vez estabelecidas as características desejáveis a um banco de dados temporais e os requisitos para obtê-las, é possível desenvolver uma nova modelagem de dados que

estenda o conceito tradicional e forneça o suporte à temporalidade. Com uma modelagem temporal, pode-se desenvolver uma linguagem específica para as consultas temporais. É possível, ainda, que um SGBD forneça todo o suporte temporal nativamente.

No capítulo seguinte, apresenta-se uma discussão aplicada sobre um modelo de dados temporal específico. Particularmente, são apresentados um novo tipo de banco de dados, o banco de dados de grafos, bem como uma justificativa para seu uso. A aplicação a este modelo dos conceitos vistos neste capítulo (que valem para os diversos tipos de bancos de dados, e não somente para o modelo ER) e um SGBD de uso comercial que trabalha com este modelo também serão discutidos.

3 BANCOS DE DADOS DE GRAFOS E MODELAGEM TEMPORAL

O capítulo anterior apresentou as principais definições formais de bancos de dados temporais. Essas definições tomaram como base a estrutura mais comum e difundida para bancos de dados: o modelo entidade-relacionamento. Discutiu-se, então, o que é desejável e necessário para um banco de dados com suporte temporal, abordando o conjunto mínimo de requisitos e o que o SGBD ou o modelo construídos devem fornecer para atendê-los.

Entretanto, excetuando-se os casos onde tratou-se explicitamente do modelo ER, todas as definições discutidas são válidas para os vários modelos de bancos de dados existentes. E o fato é que, com o passar dos anos, o modelo ER se mostrou limitado em diversos aspectos, dificultando o desenvolvimento de aplicações e suas demandas em diversas frentes.

Devido a essas limitações, segundo Leavitt (2010), um número crescente de desenvolvedores e usuários começaram a procurar diversos tipos de bancos de dados não-relacionais, frequentemente chamados NoSQL. Este autor exemplifica vantagens de modelos NoSQL sobre modelos ER, como a eficiência ao lidar com dados não estruturados, a maior facilidade de entendimento com relação ao SQL, melhor performance (particularmente importante com grandes quantidades de dados), entre outras, contrapondo-as às limitações da modelagem ER com relação a escalabilidade e complexidade.

Este trabalho usa banco de dados de grafos, uma entre várias tecnologias que seguem os princípios do NoSQL. Neste capítulo, após a apresentação de alguns trabalhos relacionados, são apresentadas as motivações para o uso deste tipo de banco de dados e da escolha do SGBD Neo4j, usado para o desenvolvimento dos capítulos seguintes. No que segue, o foco retorna à aplicação de uma modelagem temporal, de acordo com os conceitos já discutidos anteriormente, desta vez sobre um modelo de banco de dados de grafos.

3.1 Trabalhos Relacionados

Diversas referências descrevem experimentos com modelagem experimental usando estruturas de grafos. Rizzolo e Vaisman (2008) tratam o problema de modelagem e implementação de dados temporais em XML através de um grafo dirigido e rotulado. Elementos, atributos e valores são representados como nodos do grafo. Os nodos são relacionados através de arestas. A dimensão temporal é adicionada ao grafo rotulando arestas com intervalos fechados. O tempo é considerado como discreto e linearmente ordenado. O tipo de tempo considerado é o de transação, porém poderia ser substituído pelo tempo de validade de forma análoga. Rizzolo e Vaisman ainda apresentam uma linguagem de consulta temporal, denominada TXPath, e uma técnica de sumarização e indexação.

Gutierrez, Hurtado e Vaisman (2007) tratam o problema de modelagem e implementação de dados temporais em RDF através de um grafo dirigido e rotulado. Sujeitos e objetos são representados como nodos. Predicados são representados como rótulos de arestas que ligam dois nodos (um sujeito e um predicado). A dimensão temporal é adicionada ao grafo rotulando arestas com intervalos fechados ou instantes no tempo. O tempo é considerado como linearmente ordenado e discreto. O tipo de tempo considerado é o de validade, porém poderia ser substituído pelo tempo de transação de forma análoga. Gutierrez, Hurtado e Vaisman (2007) ainda apresentam uma linguagem de consulta temporal sobre o modelo proposto.

Khurana e Deshpande (2013) apresentam um índice hierárquico, distribuído, altamente configurável e extensível pelo usuário que permite compactar registros de informações históricas de relacionamentos e que suporta recuperação eficiente de *snapshots* históricos do grafo. O banco de dados utilizado é o Kyoto Cabinet, um banco de dados NoSQL que segue o modelo chave-valor. O tempo é considerado como linearmente ordenado e discreto. O tipo de tempo considerado é o de validade. Um subconjunto dos *snapshots* é explicitamente armazenado, enquanto os demais *snapshots* são representados apenas pelas suas mudanças com relação aos *snapshots* armazenados. Cada *snapshot* possui um instante de tempo associado. Khurana e Deshpande (2013) ainda apresentam uma API de programação para recuperar os *snapshots* desejados.

Entretanto, é o trabalho de Cattuto et al. (2013) a principal referência para o presente trabalho, pois se utiliza de um modelo baseado em grafo implementado em um SGBD que armazena e processa os dados nativamente como um grafo. Isso permite uma modelagem mais natural dos dados, pois a estrutura de grafos permite melhor visibilidade ao usuário; consultas podem fazer referência diretamente à estrutura do grafo, com alto grau de abstração, possuindo funções nativas para consultas do tipo menor caminho ou a subgrafos; e as consultas são localizadas para uma parte do grafo o que não degrada muito o desempenho de consultas que envolvem relacionamentos. Uma discussão mais detalhada a respeito dessas características e suas vantages é apresentada nas seções a seguir.

3.2 Contextualização e Motivações para o Uso de Banco de Dados de Grafos

Para entender a motivação do uso de banco de dados de grafos em uma aplicação, é necessário compreender a estrutura desse modelo. Angles et al. (2008) definem um banco de dados de grafos de acordo com três componentes básicos:

- *Dados e/ou esquema de dados*, representados por grafos ou estruturas de dados que generalizam a noção de grafos
- A *manipulação dos dados*, expressa através de transformações em grafos
- As *restrições de integridade*, que garantem a consistência dos dados - como por exemplo um rótulo único para um grupo de nodos do grafo.

Em resumo, um BD de grafos é um modelo no qual as estruturas de dados para o esquema e/ou instâncias são modeladas como um grafo dirigido - possivelmente rotulado - ou generalizações da estrutura de dados de um grafo, em que a manipulação de dados é expressa por operações orientadas a grafos e os tipos e restrições de integridade podem ser definidos sobre a estrutura de grafos (ANGLES; GUTIERREZ, 2008, p.5). A partir dessa breve definição, os autores justificam o uso de banco de dados de grafos por algumas de suas vantagens: permitem uma modelagem mais natural dos dados, pois a estrutura de grafos permite melhor visibilidade ao usuário; consultas podem fazer referência diretamente à estrutura do grafo, com alto grau de abstração, permitidindo facilmente consultas do tipo *menor caminho* ou a *subgrafos*; podem possibilitar estruturas de armazenamento específicas para grafos.

Robinson et al. (2013) discutem, em seu livro sobre BD de grafos, diversos aspectos que motivam o uso desta modelagem. Essas vantagens começam pela natureza da modelagem em si, que é poderosa e inovadora, mas somente questões efetivamente práticas poderiam justificar a escolha deste tipo de banco de dados sobre a já bem conhecida e estabelecida modelagem ER. Uma forte razão, então, para a escolha de um banco de dados de grafos é o absoluto aumento de desempenho quando se trata de dados conectados contra bancos de dados relacionais e outros armazenamentos NoSQL. Em contraste com os bancos de dados relacionais, onde o desempenho de consultas *join-intensivas* deteriora-se conforme o conjunto de dados se torna maior, com banco de dados de grafos o desempenho tende a permanecer relativamente constante, mesmo se conjunto de dados crescer. Isto ocorre porque as consultas são localizadas para uma parte do grafo. Como resultado, o tempo de execução para cada consulta é proporcional apenas para o tamanho da parte do grafo percorrido para satisfazer essa consulta, em vez do tamanho do grafo global (ROBINSON; WEBBER; EIFREM, 2013, p.8).

Desenvolvedores e arquitetos de dados desejam conectar dados de acordo com o domínio da aplicação, permitindo que estrutura e esquema cresçam em conjunto com o entendimento do espaço do problema, ao invés de impor essa estrutura de início, quando pouco conhecem sobre a real estrutura dos dados. Segundo Robinson et al. (2013), os bancos de dados de grafos atendem à essa necessidade diretamente. Grafos são naturalmente aditivos: é possível adicionar novos tipos de relacionamentos, nós e subgrafos à uma estrutura existente sem causar grande impacto nas consultas já existentes e, portanto, nas funcionalidades da aplicação, aumentando produtividade e diminuindo riscos. Seguindo a mesma filosofia de metodologias ágeis de desenvolvimento, não é necessário, com BDs de grafos, modelar nosso domínio em exaustão antes do tempo. A natureza aditiva dos grafos tende também a diminuir *overhead* e risco na manutenção.

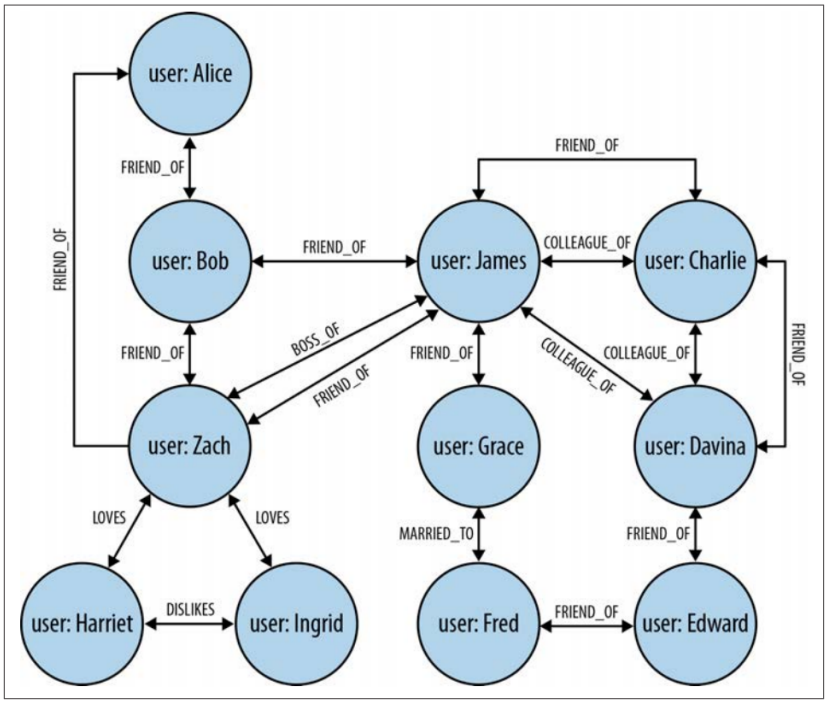
A habilidade de evoluir o modelo de dados com facilidade, de acordo a evolução de uma aplicação, também é um ponto forte da modelagem de grafos, o que está alinhado às práticas ágeis atuais de desenvolvimento de software. A natureza *schema-free* do modelo de dados de grafos, ou seja, sem um esquema que restrinja a representação dos dados a um esquema pré-definido e rígido, aliada à natureza orientada a testes da interface do SGBD, auxilia no controle da progressão de uma aplicação.

É importante ressaltar o maior problema da modelagem clássica: ironicamamente, bases de dados relacionais lidam mal com relacionamentos (ROBINSON; WEBBER; EIFREM, 2013, p.12). Os relacionamentos existem no vernáculo dos modelos ER, mas somente no contexto de unir tabelas. Relações no modelo relacional não expressam, por si só, nenhuma informação sobre a semântica dessas relações. Além disso, com o crescimento dos dados, o modelo relacional acaba deteriorando a performance com as imensas junções de tabelas, com linhas esparsamente populadas e *overhead* lógico para tratamento de *nulls*. Outros bancos de dados NoSQL, como chave/valor e orientados a documentos ou colunas, também apresentam esse problema.

Bancos de dados de grafos, entretanto, apresentam a semântica desses relacionamentos diretamente em sua estrutura. Dados conectados são armazenados como dados conectados, e não como conjuntos de informações com relacionamentos. Onde há conexões no domínio, há conexões nos dados. A Figura 3.1 é um exemplo de modelagem em grafo e mostra as caraterísticas apresentadas até aqui, pois expressa intuitivamente a semântica de dados e relacionamentos - observando que esta seria uma instância do grafo em si, e não um modelo referencial ou de documentação.

Com essas características apresentadas, pode-se inferir que a adição de suporte temporal a uma base de dados em grafo deve: ser simplificada, com base na natureza adititiva expressa por Robinson et al. (2013); deve ter pouco impacto na performance geral pois, segundo Angles et al. (2008), consultas a subgrafos são facilitadas; deve expressar com eficiência a natureza temporal das informações (e sua distinção das que não são temporais) neste grafo pois, segundo os diversos autores, a visualização da estrutura dos dados

Figura 3.1: Modelagem em Grafo de uma pequena rede social, com pessoas e seus relacionamentos



Fonte: Robinson et al. (2013).

é trivial quando os representamos em grafos.

3.2.1 O SGBD Neo4j

Existem disponíveis algumas opções, tanto abertas como proprietárias, de SGBDs de grafos. Robinson et al. (2013) consideram importante analisar dois aspectos cruciais para a escolha de um sistema para banco de dados de grafos: a maneira como os dados são armazenados e o motor de processamento destes dados. Eses autor considera que quando os dados são armazenados e processados nativamente como um grafo (ou seja, sem camadas intermediárias com um banco de dados relacional ou estratégias como serialização dos dados, por exemplo), há ganhos significativos de performance. Penteado et. al (2014) também aponta que os sistemas nativos oferecem um melhor desempenho no processamento de consultas quando comparado com os não-nativos.

Segundo Robinson et al. (2013), Neo4j é um banco de dados com capacidade de processamento de grafos nativa, bem como é nativo o armazenamento dos dados em grafos. McColl et al. (2013), em comparativo abrangendo diversos SGBDs de grafos, citam o Neo4j como um SGBD transacional, baseado em disco, com algoritmos específicos para grafos, e suporte à linguagem Cypher[1]. Cypher é uma linguagem declarativas para con-

[1]http://docs.neo4j.org/chunked/stable/cypher-query-lang.html

sultas em grafos que permite acesso e a atualização de informações no banco de dados, considerada simples mas poderosa e desenvolvida para ser humanamente legível e robusta (em contraste a linguagens como Gremlin, que também é usada para consultas em BDs de grafos)[2]. Além disso, o Neo4j é o mais popular entre os bancos de dados em uso na atualidade e tem a vantagem de transparência por ser *open source* (embora haja também uma versão comercial fechada, com mais recursos e suporte). Existe uma comunidade ativa trabalhando sobre o SGBD, com atualizações constantes (a versão mais recente disponível foi lançada em junho deste ano[3]) e documentação constante[4].

Com estas características, pode-se considerar o Neo4j o principal banco de dados de grafos entre os disponíveis, o que motiva fortemente seu uso. Não bastasse isso, a referência para a modelagem temporal do presente trabalho usou este SGBD em seus experimentos. Esta modelagem é apresentada na seção a seguir.

3.3 Modelagem Temporal com Banco de Dados de Grafos

O Capítulo 2 mostra o conceito de Banco de Dados Temporais, elucidando o que é a representação temporal e os aspectos e requisitos necessários à sua implementação. Entretanto, faz apenas a discussão conceitual, sem apresentar uma metodologia ou uma exemplificação de uma base temporal. Esse é o objetivo da presente seção, que apresenta uma possibilidade de modelagem que pode servir genericamente para qualquer tipo de problema que necessite de representação temporal. Particularmente, após a motivação apresentada na seção anterior, o objeto deste estudo são bancos de dados de grafos.

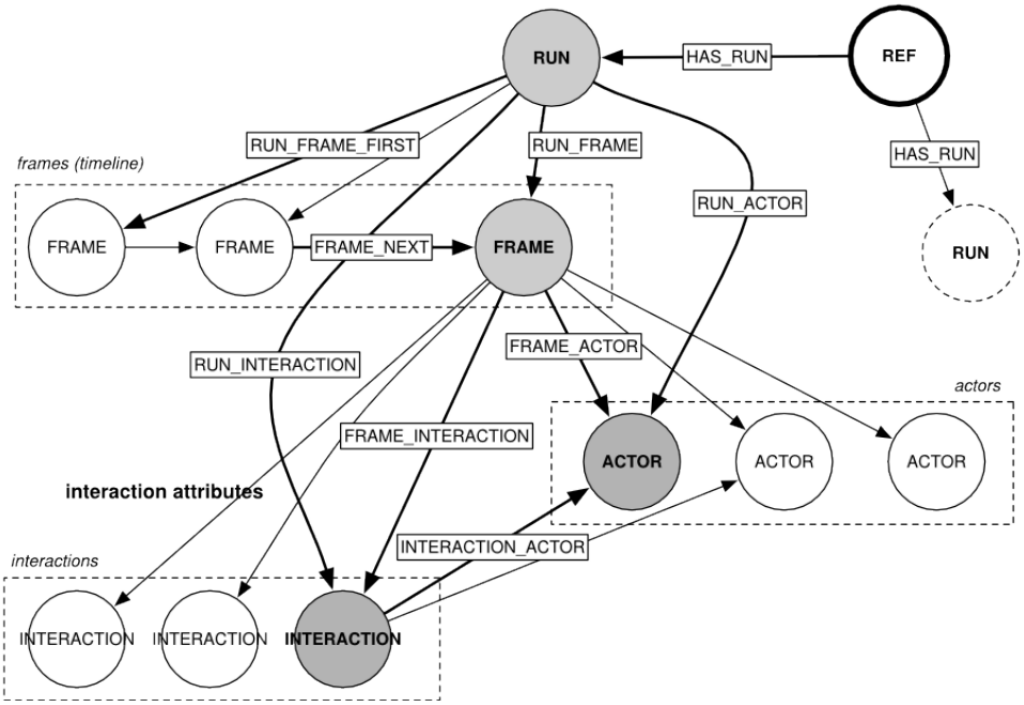
Cattuto et al. (2013) apresentam a experimentação com uma modelagem que atende aos requisitos temporais interessantes a este trabalho. Trata-se da representação das ocorrências de interações entre pessoas por proximidade, de maneira a armazenar e permitir consultas sobre estas interações ao longo do tempo. O estudo é relevante pois, em seus testes, com um conjunto consideravelmente pequeno de pessoas, 113, o conjunto de interações e consequentemente de *frames de tempo* já são bastante grandes - 2163 e 13956, respectivamente (CATTUTO et al., 2013). Para melhor compreensão da modelagem proposta, apresenta-se a seguir uma breve descrição do experimento.

O experimento de Cattuto et al. (2013) consistiu em armazenar, utilizando sensores carregados pelos participantes, todas as ocorrências de aproximação entre as pessoas. A base de dados em grafos deveria, portanto, armazenar uma identificação destes participantes, referidos daqui em diante como atores, e cada uma das ocorrências de aproximação entre eles - as interações. Assim, era necessário que fosse possível obter, dado um ponto específico no tempo, o estado de toda essa rede de proximidade entre os atores, definindo um mapa da rede social estabelecida naquele momento. Para acessar este mapa de inte-

[2] http://docs.neo4j.org/chunked/stable/cypher-introduction.html
[3] http://neo4j.com/release-notes/neo4j-2-1-2/
[4] http://neo4j.com/docs/2.1.2/

rações, é preciso registrar um nó de referência, que terá ligações com todos as diferentes fatias de tempo registradas no período do experimento, os chamados *frames*. Esses *frames* também se relacionam entre si, através de uma lista de precedência. Cada *frame* aponta, então, para um conjunto de interações - justamente as ocorridas no intervalo de tempo correspondente a este *frame*. Essas interações, por sua vez, nada mais são que a instância de uma ligação entre dois atores, ou seja, o registro de que um par de sensores esteve próximo naquele momento.

Figura 3.2: Modelagem Temporal de uma Rede Social, com indivíduos e suas interações



Fonte: Cattuto et al. (2013).

A Figura 3.2 mostra o modelo obtido por Cattuto et al. (2013) para seu experimento. O nodo *RUN* é o nodo de referência, através do qual é feito o acesso a todo o grafo nas consultas desejadas. O nodo *FRAME* representa a fatia de tempo: todos os nodos ligados a um determinado nodo *FRAME* fazem parte da rede de conexões estabelecida no período de tempo referido por este. Assim, os nodos *INTERACTION* estão ligados ao *FRAME* correspondente ao instante quando a interação ocorreu e aos atores (nodos *ACTOR*) que participaram dela. Estes nodos são a parte central do grafo e expressam o registro temporal das interações que ocorrem entre os atores: eles se ligam à "camada temporal" representada pelos nodos *FRAME* e aos nodos *ACTOR*, que formam a "camada não temporal"do grafo.

Com os dados coletados e armazenados no grafo, conforme a modelagem acima descrita, Cattuto et al. (2013) apresentam diversas consultas em linguagem Cypher, mos-

trando como é possível fazer consultas temporais aos dados, restringindo os dados consultados por um determinado ano ou dia, por exemplo. A Figura 3.3 exemplifica uma destas consultas, que obtém todos os *frames* de tempo registrado entre 09:00 e 13:00 do dia 1º de Julho de 2009, ordenando-os pelo seu *timestamp*.

Figura 3.3: Exemplo de consulta temporal em linguagem Cypher.

```
START root = node(root_node_id)
MATCH root-[:HAS_RUN]->run-[:HAS_TIMELINE]->tl,
    tl-[y:NEXT_LEVEL]->()-[m:NEXT_LEVEL]->month,
    month-[d:NEXT_LEVEL]->[h:NEXT_LEVEL]->hour,
    hour-[:TIMELINE_INSTANCE]->frame
WHERE run.name="HT2009" and y.year=2009 and m.month=7
    and d.day=1 and h.hour>=9 and h.hour<13
RETURN frame ORDER BY frame.timestamp
```

Fonte: Cattuto et al. (2013).

Cattuto et al. (2013) seguem seu experimento apresentando os resultados com medidas de performance para consulta, analisando e discutindo sua eficiência. Destaca-se o problema de degradação em escala com relação a *nodos densamente conectados*, para o qual o autor sugere revisões no modelo ou a criação de estruturas de índice adicionais. Cattuto et al. (2013) comentam também sobre relacionamentos fortemente tipados, que são relações cujo tipo expressam não só o seu significado mas incluem também informações que normalmente estão apenas em seus atributos (por exemplo, um relacionamento do tipo "ano"que tem o atributo "2014"hspace*4ptpode ser convertida para um relacionamento fortemente tipado com o tipo "ano2014"). Esse tipo de relacionamento pode evitar consultas desnecessárias às propriedades dos nodos ligados por ele, o que reduz o impacto em performance. Além disso, diversos relacionamentos podem se revelar desnecessários, e removê-los melhora a performance ao reduzir a densidade de conectividade de nodos centrais.

3.4 Considerações Finais

Bancos de dados de Grafos são uma alternativa ao modelo relacional de dados e se destacam por diversos benefícios de uma abordagem NoSQL. O armazenamento de dados extraídos da web se beneficia deste tipo de modelo por sua natureza *schema-free* e pelo seu alto poder de expressão - de maneira escalável - dos relacionamentos entre estes dados.

Através de sua análise, Cattuto et al. (2013) apresentam um modelo satisfatório, tanto em expressão como em performance, para a modelagem temporal de uma rede social com muitas interações e fatias temporais muito pequenas. Como se trata de um problema com-

plexo, vale assumir que problemas com uma granularidade de tempo e complexidade de interações entre elementos menores podem ter uma performance ainda mais satisfatória.

Com base neste estudo, este trabalho apresenta uma proposta para a modelagem temporal de um conjunto de dados que são representativos de informações obtidas na web. Nos próximos capítulos, são apresentados os dados utilizados e sua estrutura e a maneira como foram adaptados ao modelo temporal proposto por Cattuto et al. (2013), bem como, em seguida, a justificativa para esta solução, com a apresentação e discussão dos resultados dos experimentos realizados.

4 UMA PROPOSTA DE BANCO DE DADOS DE GRAFOS PARA CONSULTAS TEMPORAIS

O Capítulo 3 apresentou uma proposta para modelagem temporal usando um banco de dados de grafos. Este trabalho aplica esses conceitos para modelar um conjunto de dados do mundo real, exemplificando como o modelo proposto pode se adequar a diversos problemas práticos. No que segue, o presente capítulo descreve o processo de construção desta base de dados.

Em primeiro lugar, tem-se uma descrição dos dados e de seu pré-processamento e a preparação para a inserção em um banco de dados, já considerando as dificuldades encontradas. Com o resultado deste pré-processamento, diversas alternativas para inserção em um banco de dados de grafos são discutidas comparativamente também neste Capítulo. Por fim, apresenta-se o esquema para o banco de dados resultante, que é o objeto das consultas temporais apresentadas posteriormente.

4.1 Base de Dados Sobre Eleições Para Vereador

Diversos contextos se encaixam no problema da modelagem temporal de dados. Qualquer cenário que se altera com o tempo e precisa, de alguma maneira, armazenar essa mutação, para que se possa observar e analisar historicamente os dados pode ser modelado como um banco de dados temporal. Para este trabalho, considera-se o cenário das eleições para vereador no Brasil entre os anos de 2004 e 2012.

Essa escolha se justifica pelo fato de que dados eleitorais possuem diversos tipos de informações temporais, como o partido de filiação de candidato em determinado ano (tempo no relacionamento entre duas entidades) e o total de despesa de um candidato em determinado ano (tempo na propriedade de uma entidade). Além disso, o Tribunal Superior Eleitoral disponibiliza um repositório com dados sobre as eleições no Brasil[1]. São dados sobre o eleitorado, candidatos/candidaturas, resultados de votação e até dados sobre prestação de contas. Essa vastidão de informações validadas por uma instituição oficial e atualizadas a cada eleição permite a construção de uma base de dados relevante e que

[1] http://www.tse.jus.br/hotSites/pesquisas-eleitorais

pode ser expandida e atualizada facilmente. Para este trabalho, são considerados os dados sobre eleições municipais dos anos de 2004, 2008 e 2012. Os dados obtidos incluem informações para todos os cargos municipais, ou seja, também englobam candidatos a prefeito e vice-prefeito, mas somente as candidaturas para o cargo de vereador foram tratadas, pois compreende um escopo de dados fechado e o volume de dados é o maior entre todos os cargos e, portanto, suficiente para a solução do problema.

Os dados estão armazenados em um arquivo Zip para cada ano, contendo dados das candidaturas agrupadas por estado da federação (UF). Tem-se, assim, para o arquivo `consultas_cand_2012.zip` os arquivos `consulta_cand_2012_AC.txt`, `consulta_cand_2012_AL.txt` - onde 2012 é o ano da eleição e AC e AL são as siglas dos estados - e analogamente para os outros estados, totalizando 26 arquivos.

4.2 Inserção dos Dados

Com os dados disponíveis em arquivos CSV, é preciso fazer a leitura dos dados e um pré-processamento para classificá-los e organizá-los para a inserção no banco de dados. Para este pré-processamento (e, posteriormente, para a inserção efetiva dos dados), foi escolhida a linguagem Python[2], não só por ser uma linguagem de script de configuração inicial simples, como por ser referência em bom desempenho com arquivos e possuir um módulo para manipulação de arquivos CSV[3] nativamente integrado.

A seguir, apresenta-se as estratégias utilizadas para a criação de um script Python para pré-processamento dos dados.

4.2.1 Mapeamento e Categorização dos dados CSV

Os arquivos disponibilizados pelo TSE incluem uma documentação para referência, presente no arquivo *LEIAME.PDF*. Esse documento apresenta o esquema utilizado para cada conjunto de valores separados por vírgula, informando o que representa cada item (ou seja, coluna) em uma linha de arquivo. A partir dessa documentação é possível classificar e organizar os dados.

Cada linha de um arquivo CSV contém, portanto, diversos tipos de informações a respeito de uma candidatura. Algumas delas dizem repeito somente à candidatura em questão (por exemplo, a coligação pela qual um candidato concorre e o seu número de candidatura); outras, são informações temporais do candidato, referentes ao ano corrente (estado civil e escolaridade, por exemplo); outras, são a respeito de um candidato, que não se alteram de uma eleição para outra (como, por exemplo, número de seu título eleitoral). Além disso, uma linha do CSV contém dados organizacionais que precisam ser armazeados, como o município da candidatura, e outros que podem ser descartados, como códigos

[2]https://www.python.org/

[3]https://docs.python.org/2/library/csv.html

de controle do TSE. A Figura 4.1 mostra o mapeamento do esquema disponibilizado no manual do TSE para o esquema criado e utilização nos scripts de inserção.

Figura 4.1: Descrição dos valores de cada coluna, conforme leiaute disponibilizado pelo TSE (à esquerda) e mapeamento de coluna do CSV para indexação por tabela associativa (à direita). A tabela não contempla dados do CSV que não foram utilizados.

Descrição dos valores do arquivo CSV, conforme tabela disponibilizada pelo TSE	
Variável	**Descrição**
ANO_ELEICAO	Ano da eleição
SIGLA_UF	Sigla da UF da eleição
SIGLA_UE	Sigla da Unidade Eleitoral
DESCRICAO_UE	Descrição da Unidade Eleitoral
NOME_CANDIDATO	Nome completo do candidato
NUMERO_CANDIDATO	Número do candidato na urna
NOME_URNA_CANDIDATO	Nome de urna do candidato
DES_SITUACAO_CANDIDATURA	Descrição da situação de candidatura
NUMERO_PARTIDO	Número do partido
SIGLA_PARTIDO	Sigla do partido
NOME_PARTIDO	Nome do partido
COMPOSICAO_LEGENDA	Composição da legenda
NOME_LEGENDA	Nome da legenda
DESCRICAO_OCUPACAO	Descrição da ocupação
DATA_NASCIMENTO	Data de nascimento do candidato
NUM_TITULO_ELEITORAL_CANDIDATO	Número do título eleitoral
IDADE_DATA_ELEICAO	Idade do candidato da data da eleição
DESCRICAO_SEXO	Descrição do sexo do candidato
COD_GRAU_INSTRUCAO	Código do grau de instrução
DESCRICAO_GRAU_INSTRUCAO	Descrição do grau de instrução
DESCRICAO_ESTADO_CIVIL	Descrição do estado civil
DESCRICAO_NACIONALIDADE	Descrição da nacionalidade
DESPESA_MAX_CAMPANHA	Despesa máxima de campanha
DESC_SIT_TOT_TURNO	Descrição da situação de totalização

Esquema para mapear colunas de um CSV *para um dictionary* em Python		
Coluna	**Índice ***	**Chave**
3	A	ANO
6	E	UF
7	S	SIGLA_UE
8	C	NOME
11	F	NOME
13	CA	NUMERO
14	CA	NOME_URNA
16	CA	SITUACAO
17	P	NUMERO
18	P	SIGLA
19	P	NOME_PARTIDO
22	CO	COMPOSICAO
23	CO	NOME
25	T	OCUPACAO
26	F	DATA_NASC
27	F	NUM_TITULO
28	T	IDADE
30	F	SEXO
31	T	COD_GRAU_INSTRUCAO
32	T	GRAU_INSTRUCAO
34	T	ESTADO_CIVIL
36	F	NACIONALIDADE
40	CA	DESPESA_MAX_CAMPANHA
42	CA	SIT_TOT_TURNO

*Legenda: **A**: Ano; **E**: Estado; **C**: Cidade; **CA**: Candidatura; **F**: Dados Candidato; **T**: Dados Temporais; **CO**: Coligação; **P**: Partido; **S**: Dado auxiliar (não armazenado)

Fonte: Elaborado pelo autor.

Cada linha do CSV é processada e então mapeada utilizando a estrutura *dictionary*, que permite a criação de tabelas associativas (*chave-valor*) para a organização dos dados. Assim, após seu processamento, cada linha fica armazenada em uma estrutura organizada com o esquema de referência mostrado na tabela à direita da Figura 4.1. Por exemplo: considerando uma variável do tipo *dictionary* chamada `linhaCSV`, pode-se obter o estado civil do candidato acessando `linhaCSV['T']['estado_civil']`. Com essa estratégia, foi possível modularizar, de certa forma, o pré-processamento. Diversas alternativas de inserção dos dados podem ser experimentadas, pois basta extrair a informação da variável `linhaCSV`; além disso, normalizações e tratamentos podem ser facilmente

tratadas sem que a inserção seja necessariamente afetada.

4.2.2 Alternativas Para Inserção dos Dados

Para este trabalho, diversas alternativas para a inserção dos dados foram consideradas. De fato, estratégias diferentes para inserção foram implementadas e preteridas devido a problemas principalmente de desempenho. A seguir, apresenta-se a análise essas alternativas.

Inserção por consultas Cypher: a linguagem Cypher é a linguagem de referência e proprietária do SGBD Neo4j[4]. Cada linha do CSV pode ser convertida em um conjunto de primitivas `CREATE` ou `MERGE`, que inserem os dados (no caso do `MERGE`, verificando se já existem).

Essas consultas são geradas para um conjunto de linhas e enviadas, utilizando o módulo Python *GraphDatabase*, uma classe que implementa o *Rest Client para Neo4j*[5] - cliente que aceita requisições `HTTP POST` e as envia para o banco de dados. A utilização deste cliente implica em balancear as consultas para minimizar o *overhead* de rede, enviando consultas em lote. Esta alternativa demanda a escolha de um o tamanho ótimo para estes lotes: não muito grandes que o SGBD tenha problemas para tratar e não muito pequenas que gerem muito *overhead* de requisições. Assim, são enviadas consultas como as apresentadas na Figura 4.2, que mostra as inserções, usando `CREATE` e `MERGE`, para uma candidatura (ou uma linha do CSV).

Figura 4.2: Consultas Cypher para inserção no banco de de dados

```
MATCH (m:Eleicoes)
MERGE (ano2004:Ano { ano: 2004 })
MERGE (m)-[:ano2004]->(ano2004)
MERGE (mun_e7:Cidade { id_unico: "ACACRELANDIA" })SET mun_e7.nome="ACRELANDIA"
MERGE (est_e7:Estado { uf:"AC" })
MERGE (mun_e7)-[r_e7:pertence]->(est_e7)
MERGE (c7:Candidato { num_titulo: 1022542470 })SET c7.nome="ALAOR RODRIGUES",c7.sexo="MASCULINO",c7.nacionalidade="BRASILEIRA NATA"
CREATE (dados_t7:DadosTemp { ocupacao:"OUTROS", idade:"-1", grau_instrucao:"LÊ E ESCREVE", estado_civil:"CASADO(A)" })
CREATE (c7)-[vinc_t7:dados_temp]->(dados_t7)
MERGE (partido7:Partido { sigla : "PMDB"})SET partido7.sigla="PMDB",partido7.nome_partido="PARTIDO DO MOVIMENTO DEMOCRÁTICO BRASILEIRO"
MERGE (coligacao7:Coligacao { id_unico : "2004AC1120CONTINUANDO O PROGRESSO II"})SET coligacao7.nome="CONTINUANDO O PROGRESSO II"
CREATE (candidatura7:Candidatura { nome_urna:"ALAOR", situacao:"DEFERIDO", sit_tot_turno:"SUPLENTE" })
CREATE (candidatura7)-[vinc_ca_cand7:ref_candidato]->(c7)
CREATE (candidatura7)-[vinc_ca_dados7:dados_candidato]->(dados_t7)
CREATE (candidatura7)-[vinc_ca_col7:pela_coligacao]->(coligacao7)
CREATE (candidatura7)-[vinc_ca_par7:partido_candidato]->(partido7)
CREATE (ano2004)-[vinc_ano_cand7:eleicao_candidato]->(candidatura7)
MERGE (pt_c7_1:Partido { sigla : "PP" })
MERGE (pt_c7_2:Partido { sigla : "PMDB" })
MERGE (pt_c7_3:Partido { sigla : "PPS" })
MERGE (pt_c7_4:Partido { sigla : "PFL" })
MERGE (coligacao7)-[vinc_part_col7_1:agrega]->(pt_c7_1)
MERGE (coligacao7)-[vinc_part_col7_2:agrega]->(pt_c7_2)
MERGE (coligacao7)-[vinc_part_col7_3:agrega]->(pt_c7_3)
MERGE (coligacao7)-[vinc_part_col7_4:agrega]->(pt_c7_4)
MERGE (coligacao7)-[vinc_col_cid7:municipio]->(mun_e7)
```

Fonte: Elaborado pelo autor.

O `CREATE` pode ser utilizado para dados que são referenciados uma única vez. O `MERGE`, entretanto, é necessário pois, como no exemplo acima, seria necessário refe-

[4]http://www.neo4j.org/learn/cypher
[5]https://neo4j-rest-client.readthedocs.org

renciar `ano2004` e `mun_e7` várias vezes. Manter a variável que aponta para um nodo criado em memória até o fim da inserção não seria possível, pois a memória necessária extrapolaria a do hardware disponível. Além disso, ao enviar requisições em lote, seria necessário obter essa variável novamente de qualquer maneira. A grande vantagem deste método é que, com a primitiva `MERGE`, é possível deixar que o SGDB cuide de questões de unicidade. Por exemplo, para o município, o atributo `id_unico` com uma restrição de unicidade garante que existirá um único nó com a cidade Acrelândia pertencente ao estado do Acre. Essa característica implica em complexidade exponencial para as inserções, uma vez que, com mais dados inseridos, consultas `MERGE` se tornam cada vez mais lentas. Além disso, o cliente se revela pouco eficiente na execução destas consultas, mesmo em testes apenas usando consulta `CREATE`.

Inserção Usando o Módulo *GraphDatabase*: na abordagem anterior, o módulo *GraphDatabase* foi utilizado apenas para enviar as consultas Cypher ao servidor. Entretanto, ele possui diversos métodos próprios para manipulação do banco de dados, tanto para consultas como para inserções. A utilização desses métodos também foi experimentada. Com essa ferramenta, a inserção de um nodo foi simplificada, mas outros controles passaram a ser necessários. A criação de um nodo e de um relacionamento possui o formato apresentado pelo código Python da Figura 4.3, que mostra inserções de nodos para Cidade e Estado da candidatura.

Figura 4.3: Código Python para inserções usando Rest Client (módulo GraphDatabase)

```
######  CIDADE  E  ESTADO  DA  CANDIDATURA  ########

UF = tabelas['E']['uf']

if UF not in lista_inseridos['estados']:

    node_estado = gdb.nodes(uf = UF)
    lista_labels['estados'][iter_estado] = node_estado
    iter_estado = iter_estado + 1
    lista_inseridos['estados'][UF] = node_estado
else:
    node_estado =lista_inseridos['estados'][UF]

if id_unic_cid not in lista_inseridos['cidades']:

    node_cidade = gdb.nodes(nome = tabelas['C']['nome'])
    lista_labels['cidades'][iter_cidade] = node_cidade
    iter_cidade = iter_cidade+1
    lista_inseridos['cidades'][id_unic_cid] = node_cidade
    node_cidade.relationships.create("pertence", node_estado)
else:
    node_cidade = lista_inseridos['cidades'][id_unic_cid]
##############################################
```

Fonte: Elaborado pelo autor.

Apesar dessa simplificação em relação às consultas Cypher, não existe a possibilidade de deixar a tarefa de tratamento de dados únicos para o SGBD, como na consulta do tipo `MERGE`. Se antes, por exemplo, o identificador de cidade única era simplesmente enviado para a consulta, agora ele precisa ser tratado; assim, se uma determinada cidade não existe, ela é criada e um apontador para o seu dono é armazenado em uma lista - uma variável *dictionary* em Python - e, se já existe, ele é encontrado na lista das cidades existentes. O fato deste tratamento ser feito sobre estas listas em Python acelerou a execução das requisições enviadas ao Neo4j - que, também neste caso, eram agrupadas e enviadas em lote (neste caso, com abertura de transações e *commits*). Apesar disso, por consequência dos tratamentos de unicidade, o código ficou mais complexo e o tempo de execução do script Python aumentou. Os resultados mostraram que esta alternativa é, portanto, melhor que a anterior, mas que também tem problemas. O armazenamento dos nodos já criados em variáveis complexas do módulo *GraphDatabase* e a necessidade de mantê-las até o fim da execução aumentou o uso de memória pelo Python durante a execução e, mesmo após otimizações e descarte de alguns dados, ainda foi significante. Além disso, a complexidade exponencial, que antes era um problema para o SGBD, acabou apenas trocando de lugar, sendo transferida para o script Python.

Essa foi a alternativa utilizada por Cattuto et al. (2013) para inserção dos dados[6], no trabalho apresentado no Capítuto 3. Entretanto, trata-se de um volume de dados muito menor do que este trabalho aborda.

Inserção Usando *Geoff*: o formato *Geoff*[7] é um formato para expressão de dados que possuem a forma de grafos. Compreensível para um leitor que conhece a sintaxe da linguagem Cypher, como mostra a Figura 4.4, este formato foi considerado por existir uma extensão para o Neo4j capaz de inserir grandes volumes de dados expressos nessa forma. O plugin *Load2Neo*[8] apresenta, em sua página, tabelas com tempos de execução que, no cenário ideal, seriam muito baixos.

De fato, os tempos observados utilizando esta alternativa para um estado específico, ou seja, construindo todo o grafo para um estado no formato *Geoff* e executando sua inserção, foram inferiores aos obtidos com as alternativas já descritas. Entretanto, o problema mencionado para o `MERGE` das consultas Cypher volta a aparecer aqui. É necessário ter uma referência para os nodos já inseridos e isso é possível com o formato *Geoff*, com uma primitiva que muito se assemelha com o `MERGE` em Cypher, necessária ao inserir os dados em lote. Uma solução para evitar esse problema seria, portanto, criar um único arquivo *Geoff* com os dados de todos os estados, o que, em tese, funcionaria sem problemas, pois não é apresentado nenhum impedimento teórico na descrição da extensão *Load2Neo*. Na prática, porém, observou-se que isso não funciona: mesmo dividindo as

[6] https://github.com/SocioPatterns/neo4j-dynagraph/blob/master/load_gexf_to_neo4j.py
[7] http://nigelsmall.com/geoff
[8] http://nigelsmall.com/load2neo

Figura 4.4: Grafo expresso no formato Geoff

```
(eleicoes:Eleicoes!nome { "nome": "Eleições"})
(ano2004:Ano!ano  { "ano" : 2004})
(eleicoes)-[:2004]->(ano2004)
(ufAC:Estado { "uf": "AC" })
(ACRELANDIAAC:Cidade {"id_unic_cid": "ACRELANDIAAC", "nome": "ACRELANDIA"})
(ACRELANDIAAC)-[:pertence]->(ufAC)
(partPMDB:Partido {"sigla": "PMDB", "numero": "15", "nome_partido": "PARTIDO DO MOVIMENTO DEMOCRÁTICO BRASILEIRO"})
(200401120ACCONTINUANDOOPROGRESSOII:Coligacao {"nome": "CONTINUANDO O PROGRESSO II", "id_unic_col": "200401120ACCONTINUANDOOPROGRESSOII"})
(200401120ACCONTINUANDOOPROGRESSOII)-[:pelo_municipio]->(ACRELANDIAAC)
(partPP:Partido { "sigla" : "PP" })
(200401120ACCONTINUANDOOPROGRESSOII)-[:agrega]->(partPP)
(200401120ACCONTINUANDOOPROGRESSOII)-[:agrega]->(partPMDB)
(partPPS:Partido { "sigla" : "PPS" })
(200401120ACCONTINUANDOOPROGRESSOII)-[:agrega]->(partPPS)
(partPFL:Partido { "sigla" : "PFL" })
(200401120ACCONTINUANDOOPROGRESSOII)-[:agrega]->(partPFL)
(001022542470:Candidato {"num_titulo": "001022542470", "nacionalidade": "BRASILEIRA NATA", "data_nasc": "19011957", "nome": "ALAOR RODRIGUES", "sexo":
"MASCULINO"})
(temp0010225424702004:DadosTemp {"ocupacao": "OUTROS", "grau_instrucao": "LÊ E ESCREVE", "idade": "-1", "cod_grau_instrucao": "2", "estado_civil":
"CASADO(A)"})
(cand0010225424702004:Candidatura {"situacao": "DEFERIDO", "numero": "15120", "sit_tot_turno": "SUPLENTE", "despesa_max_campanha": "5000", "nome_urna":
"ALAOR"})
(ano2004)-[:ano_candidatura]->(cand0010225424702004)
(cand0010225424702004)-[:dados_temp]->(temp0010225424702004)
(cand0010225424702004)-[:ref_candidato]->(001022542470)
(cand0010225424702004)-[:pela_coligacao]->(200401120ACCONTINUANDOOPROGRESSOII)
(cand0010225424702004)-[:pelo_partido]->(partPMDB)
(cand0010225424702004)-[:pela_cidade]->(ACRELANDIAAC)
```

Fonte: Elaborado pelo autor.

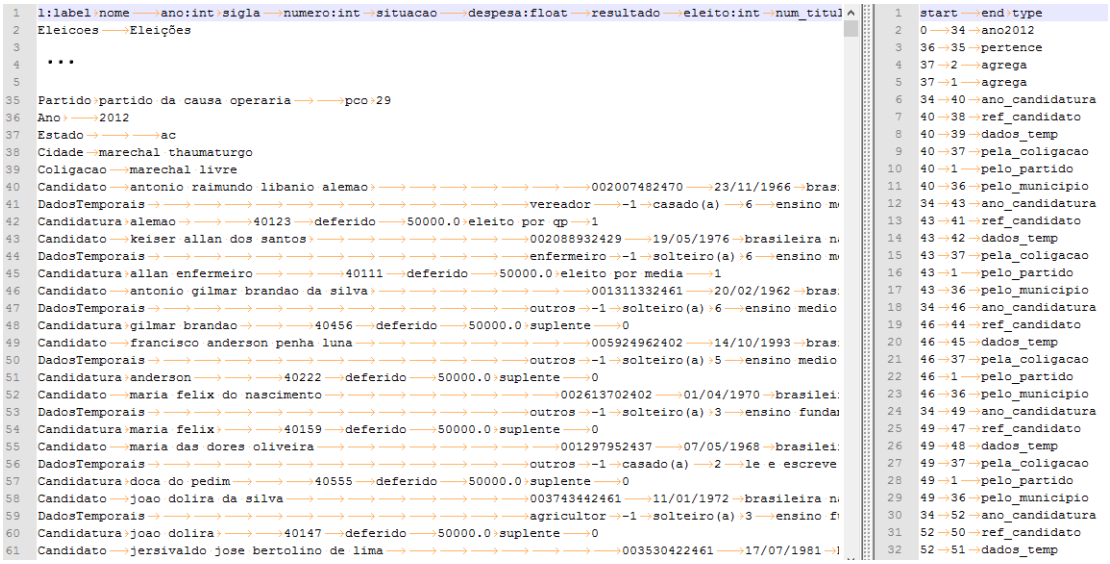
inserções em grupos por estado, arquivos gerados que eram muitos grandes, após envio para o *Load2Neo*, não produziam nenhuma inserção. A inserção em lote foi novamente utilizada e o "problema do `MERGE`"ocorreu também neste caso. Apesar dos problemas, esta foi a única inserção executada para todos os anos e todos os estados completada, entre as alternativas já descritas. O tempo de execução, entretanto, passou de três dias. Esses tempos são inviáveis, uma vez que inserções de grandes volumes de dados podem ser frequentes (como na inclusão das candidaturas de um novo ano, por exemplo). Problemas com tratamentos de inserção única e *overhead* de requisições de redes, além da complexidade exponencial como um gargalo para o SGBD motivaram a busca de outra solução.

Inserção Usando *Neo4j (CSV) Batch Importer*[9]: esta alternativa, que corresponde a uma implementação independente em Java de um importador de dados, possui uma importante característica, que motivou o seu uso: ela não depende do SGBD para sua execução, pois monta, a partir de entradas no formato CSV para nodos e relacionamentos, toda a estrutura de arquivos de um banco de dados Neo4j. A partir de um par de arquivos CSV (no formato exemplificado pela Figura 4.5, que mostra trechos de arquivos de nodos e relacionamentos), a base de dados é criada por completo, sendo um problema apenas de leitura e escrita de dados em disco. Isto elimina completamente o *overhead* de requisições ao SGBD, que sequer está envolvido no processo.

A maior parte do processamento utilizando esta solução está no script Python, que precisa fazer todo e qualquer tratamento e o armazenamento informações já inseridas, quando for necessário referenciá-las posteriormente. Só é possível inserir nodos uma única vez (cada linha do arquivo CSV para nodos representa um nodo inserido), e nunca

[9]https://github.com/jexp/batch-import

Figura 4.5: Arquivos CSV para inserção usando Batch Importer (à esquerda, linhas de um arquivo CSV com nodos; à direita, linhas de um arquivo com relacionamentos (com *ID* dos nodos origem e destino)



```
l:label;nome;ano:int;sigla;numero:int;situacao;despesa:float;resultado;eleito:int;num_titul
Eleicoes;Eleições
...
Partido;partido da causa operaria;;;pco;29
Ano;;;2012
Estado;;;;ac
Cidade;marechal thaumaturgo
Coligacao;marechal livre
Candidato;antonio raimundo libanio alemao;;;;;;;;002007482470;23/11/1966;bras
DadosTemporais;;;;;;;;;;;;;vereador;;-1;casado(a);6;ensino m
Candidatura;alemao;;;40123;deferido;50000.0;eleito por qp;1
Candidato;keiser allan dos santos;;;;;;;;002088932429;19/05/1976;brasileira n
DadosTemporais;;;;;;;;;;;;;enfermeiro;-1;solteiro(a);6;ensino m
Candidatura;allan enfermeiro;;;40111;deferido;50000.0;eleito por media;1
Candidato;antonio gilmar brandao da silva;;;;;;;;001311332461;20/02/1962;bras
DadosTemporais;;;;;;;;;;;;;outros;-1;solteiro(a);6;ensino medio
Candidatura;gilmar brandao;;;40456;deferido;50000.0;suplente;0
Candidato;francisco anderson penha luna;;;;;;;;005924962402;14/10/1993;bras
DadosTemporais;;;;;;;;;;;;;outros;-1;solteiro(a);5;ensino medio
Candidatura;anderson;;;40222;deferido;50000.0;suplente;0
Candidato;maria felix do nascimento;;;;;;;;002613702402;01/04/1970;brasilei
DadosTemporais;;;;;;;;;;;;;outros;-1;solteiro(a);3;ensino funda
Candidatura;maria felix;;;40159;deferido;50000.0;suplente;0
Candidato;maria das dores oliveira;;;;;;;;001297952437;07/05/1968;brasilei
DadosTemporais;;;;;;;;;;;;;outros;-1;casado(a);2;le e escreve
Candidatura;doca do pedim;;;40555;deferido;50000.0;suplente;0
Candidato;joao dolira da silva;;;;;;;;003743442461;11/01/1972;brasileira n
DadosTemporais;;;;;;;;;;;;;agricultor;-1;solteiro(a);3;ensino f
Candidatura;joao dolira;;;40147;deferido;50000.0;suplente;0
Candidato;jersivaldo jose bertolino de lima;;;;;;;;003530422461;17/07/1981;
```

```
start;end;type
0;34;ano2012
36;35;pertence
37;2;agrega
37;1;agrega
34;40;ano_candidatura
40;38;ref_candidato
40;39;dados_temp
40;37;pela_coligacao
40;1;pelo_partido
40;36;pelo_municipio
34;43;ano_candidatura
43;41;ref_candidato
43;42;dados_temp
43;37;pela_coligacao
43;1;pelo_partido
43;36;pelo_municipio
34;46;ano_candidatura
46;44;ref_candidato
46;45;dados_temp
46;37;pela_coligacao
46;1;pelo_partido
46;36;pelo_municipio
34;49;ano_candidatura
49;47;ref_candidato
49;48;dados_temp
49;37;pela_coligacao
49;1;pelo_partido
49;36;pelo_municipio
34;52;ano_candidatura
52;50;ref_candidato
52;51;dados_temp
```

Fonte: Elaborado pelo autor.

atualizá-los, o que pode implicar em mais um um pré-processamento. Entretanto, para referenciar nodos já tratados, basta armazenar o `ID` numérico correspondente ao nodo, e só incluí-lo na inserção após o pré-processamento total. Para este trabalho, foi necessário tratar previamente todas as informações sobre os partidos políticos, pois um partido aparece em dois pontos do CSV: primeiro, como o partido do candidato, com suas informações completas (nome, número, e sigla) e outra vez nas coligações, onde somente a sigla é informada, o que permite que um partido apareça em uma coligação antes de ter sido inserido com seus dados completos. O mesmo tratamento poderia ser necessário para candidatos que aparecem mais de uma vez, no que diz respeito às informações não temporais do candidato, mas optou-se, por simplicidade, armazenar o dado mais recente (pois dados como sexo, data de nascimento ou nome do candidato não se alteram, a menos de raras exceções).

Uma limitação importante desta aplicação é que, por gerar a base dos dados a partir do zero e sem o servidor em execução, ela é ideal somente para uma base de dados incial. Existe a opção de manter a base existente na pasta indicada como parâmetro ao importador, mas esta opção é experimental (em testes executados, funcionou sem problemas). Além disso, como todos os relacionamentos são definidos utilizando os *IDs* dos nodos origem e destino, a manipulação de uma base de dados existente seria extremamente complicada, ou teria de ser feito um armazenamento prévio destes valores. Entretanto, os tempos encontrados foram extremamente inferiores a todas as outras alternativas. Ao

surgir a necessidade de adicionar novos dados a uma base existente, por exemplo, pode ser mais interessante reconstruir a base de dados por completo, usando esta alternativa, do que adicionar um volume grande de dados com primitivas Cypher. Esta foi a alternativa adotada durante todo o trabalho. Para fins de depuração, foi mantida a inserção em lote: divindo a inserção por cada estado, problemas podem ser mais facilmente encontrados. De fato, houve erros provocados por ruídos nos dados em arquivos específicos e sua detecção seria muito mais difícil em um arquivo que compreendesse todas as inserções.

A Tabela 4.1 apresenta os tempos de execução das alternativas descritas, que justificam a escolha do último método. Neste caso, foram considerados dados dos três anos, mas de apenas três estados (AC, AL e AM). Juntos, os arquivos correspondentes somam 20,7 MiB, e consistem em apenas 3,74% do volume de dados total modelado neste trabalho (553 MiB). Esta comparação, portanto, não explicita o problema exponencial, mais relevante em algumas inserções do que em outras. A magnitude das diferenças observadas, porém, explica o preterimento das primeiras alternativas (apesar da maior dificuldade de implementação do último método).

Tabela 4.1: Tempos de execução para diferentes estratégias de inserção (parcial, considerando apenas dados dos estados AC, AL e AM)

Estratégia de inserção	Tempo de execução (em segundos)
Consultas Cypher	5573,62
Módulo GraphDatabase (RestClient)	1045,03
Geoff com Load2Neo	412,47
CSV com Batch Importer	39,94

Fonte: Elaborado pelo autor.

4.2.3 Pré-processamento, Ruídos e Normalizações

Durante todo o processo de inserção, ao longo das várias alternativas experimentadas, diversos ruídos foram encontrados no pré-processamento dos dados. A seguir, lista-se alguns deles e como foram tratados:

- **Codificação:** os arquivos disponibilizados pelo TSE estão em formato ANSI; optou-se por normalizar a codificação para UTF-8 em todo o processo, convertendo os arquivos de origem, tratando os scripts Python sempre nesta codificação, o que resulta em um banco de dados normalizado e sem conflitos;

- **Formato CSV:** o formato definido para os arquivos é de valores informados entre aspas duplas e separados por ponto-e-vírgula. Este formato, entretanto, não é seguido à risca e gera problemas em importações. Por isso, foi necessário eliminar problemas, como aspas duplas e ponto-e-vírgula contidas dentro de um valor.

Uma expressão regular que substitui esses valores - aspas duplas por aspas simples e ponto-e-vírgula por vírgula - executada para todos os arquivos com o auxílio do editor Notepadd++[10], resolveu o problema.

- **Caracteres especiais:** discrepâncias nas informações sobre as candidaturas devido a caracteres especiais, precisaram ser tratadas. Por exemplo: um candidato que se candidatou em dois anos diferentes, pode cadastrar seu nome uma vez com acento e outra sem. Municípios também apareceram com cadastros em conflito. Para resolver isso, adotou-se a normalização descrita por Manning et al. (2008), removendo caracteres especiais e salvando todos os dados em *lowercase*, o que o autor justifica pelo fato de os usuários costumarem ignorar esses detalhes ao fazer buscas, digitando os valores da maneira mais simplificada possível.

- **Erros nos dados informados:** diversos problemas com dados incompletos ou mal formatados apareceram. Candidatos sem informação de título de eleitor obrigaram a utilizar não só o título como também o nome como "chave única"do candidato.

- **Dados não informados:** a importação considerou a idade do candidato na candidatura, uma vez que essa informação aparece no manual disponibilizado pelo TSE. Porém, após a inserção, observou-se o valor `-1` para este campo. O atributo foi mantido, mas não tem utilidade, sendo impossível fazer consultas que o considerem. Em trabalhos futuros, este valor pode ser calculado considerando a data de nascimento do candidato, que está informada corretamente para a maioria dos registros, e a data do pleito corrente (que precisa ser pesquisada; ou, então, poderia-se considerar a data de registro dos dados para a idade do candidato naquela eleição).

- **Discrepância entre informações a cada ano:** observou-se que o a informação sobre o resultado da candidatura mudou de padrão ao longo do tempo. Para eleições dos anos 2004 e 2008, pode-se concluir que um candidato foi eleito apenas testando se a chave corresponde a "ELEITO". Entretanto, a partir de 2012, esta chave assume valores como "ELEITO POR QP"e "ELEITO POR MÉDIA". Por isso, para facilitar as consultas posteriormente, já que este é um dado crítico, esta informação foi pré-processada e a informação de eleito/não-eleito ficou como um novo atributo da candidatura.

De maneira geral, essas medidas mitigaram a maioria dos problemas com a manipulação e importação dos dados. Sugere-se, para trabalhos futuros, observar estas mesmas normalizações.

[10] http://notepad-plus-plus.org/

4.3 Modelo de Dados Resultante

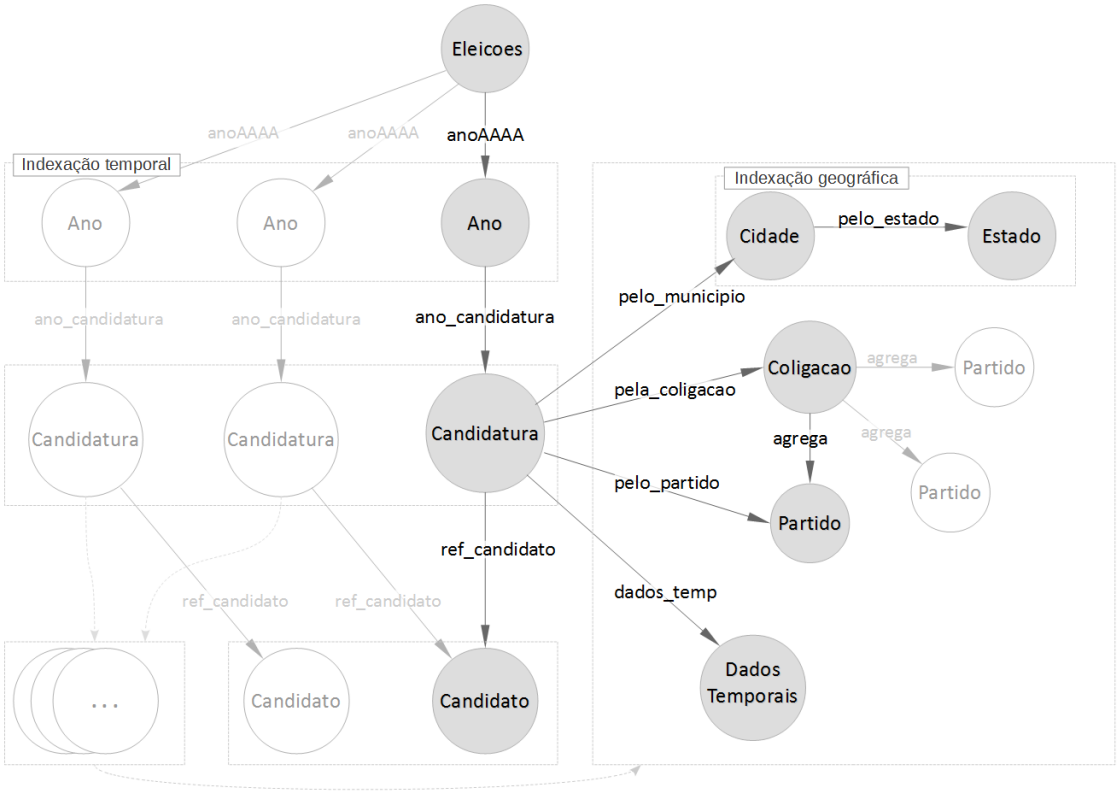
Com os dados tratados, categorizados e normalizados, conforme seção anterior, foi possível trabalhar na modelagem dos dados para possibilitar as consultas temporais. Esta modelagem segue o modelo proposto por Cattuto et al. (2013), conforme Figura 3.2, para modelagem temporal em banco de dados de grafos. A modelagem para o caso deste trabalho é, entretanto, muito mais simplificada. A indexação temporal se restringe a um nodo, pois a granularidade temporal sendo considerada é de apenas um ano. Ou seja, como não existe uma hierarquia na unidade de tempo, apenas nodos para indexação dos anos são necessárias. Eleições para vereador ocorrem uma vez a cada quatro anos e apenas em um turno. Se considerados outros pleitos poderia-se ter, por exemplo, a hierarquia ano - turno, o que alteraria este modelo.

O modelo proposto inclui as seguintes entidades principais: Candidato, Partido e Coligação. Uma instância de um candidato em uma determinada eleição é manifestada pela entidade Candidatura, que faz a conexão do candidato com os seus nesta determinada instância. Tem-se dois tipos de informação temporal. O primeiro é o relacionamento temporal entre duas entidades, sendo representadas pelo relacionamento do Candidato com Partido (por meio do nodo Candidatura) e de Partido com Coligação. O segundo é o relacionamento entre o Candidato e sua propriedades temporais, representado pelo relacionamento entre um Candidato e seus Dados Temporais, que contém os atributos escolaridade, estado civil e idade, que são informações específicas do Candidato mas válidas somente para o ano da Candidatura em questão. A Candidatura também armazena atributos temporais: nome de urna do candidato, número de urna da candidatura, despesa declarada, situação e resultado da candidatura.

O modelo é apresentado na Figura 4.6 e contém os seguintes elementos:

- O grafo, como um todo, é acessado pelo nodo referência `Eleicoes`.
- O nodo `Eleicoes` se conecta a todos os nodos `Ano` , por meio de um relacionamento tipado pelo ano em questão: `ano2004`, `ano2008` e `ano2012` (uma breve discussão sobre relacionamentos fortemente tipados é apresentada ao final deste trabalho). Os nodos `Ano` correspondem aos nodos `FRAME` do modelo de referência. Estes nodos tem o atributo `ano`, úteis para comparar consultas utilizando o relacionamento fortemente tipada como filtro, bem como para selecionar intervalos que compreendem vários anos.
- Os nodos `Ano` se conectam aos nodos `Candidatura` por meio do relacionamento `ano_candidatura`. Os nodos `Candidatura` contém também todos os atributos referentes à candidatura, como nome de urna do candidato, número de urna da candidatura e o resultado (se eleito ou não), entre outros. Estes nodos podem ser comparados ao nodo `INTERACTION` do modelo de referência.

Figura 4.6: Esquema de dados criado para o banco de dados de grafos.



Fonte: Elaborado pelo autor.

- Nodos `Candidatura` se conectam aos nodos `Partido` por meio do relacionamento `pelo_partido`, que detona o partido pelo qual o candidato concorreu. Estes nodos contém atributos como o nome, a sigla e o número do partido.
- Nodos `Candidatura` se conectam também aos nodos `Coligacao` por meio do relacionamento `pela_coligacao`, que detona a coligação pelo qual o candidato concorreu.
- Nodos `Coligacao` se conectam aos partidos por meio do relacionamento `agrega`, que detona os partidos envolvidos em uma coligação específica.
- Nodos `Candidatura` se conectam aos nodos `Cidade` por meio do relacionamento `pelo_municipio`, que representa o municípcio onde foi registrada a candidatura.
- Nodos `Cidade` se conectam aos nodos `Estado` por meio do relacionamentoo `pertence`, que representa o estado ao qual o município pertence.

- Nodos `Coligacao` se conectam aos nodos `Candidato` por meio do relacionamento `ref_candidato`, que representa o candidato referente a uma candidatura. Nodos `Candidato` contém atributos para todos os dados fixos do candidato, ou seja, dados que não se alteram com o tempo (ou pelo menos assim considerados neste trabalho, por simplicidade, já que, na prática, nome e data de nascimento de pessoas físicas podem ser alteradas). Estes nodos podem ser equiparados aos nodos `ACTOR` do modelo de referência.

- Por fim, nodos `Coligacao` se conectam ainda aos nodos `DadosTemporais` por meio do relacionamento `dados_temp`, expressando a ligação de um candidato com os dados pessoais que foram considerados temporais, como estado civil e escolaridade, entre outros.

Este modelo possui duas estruturas de indexação: uma temporal (ano) e uma geográfica (cidade –> estado), a qual é hierárquica. Quanto ao aspecto temporal, seguem algumas características importantes:

- **Ordem no tempo:** tempo totalmente ordenado e absoluto.

- **Variação temporal:** representação discreta.

- **Granularidade temporal:** ano.

- **Representação do tempo:** instante no tempo.

- **Tipo de tempo:** tempo de validade.

Este estudo de caso, portanto, possui incompletude de informações, uma vez que apenas se conhece o valor dos atributos temporais em um ponto a cada quatro anos. Não é possível inferir, por exemplo, qual foi o ponto exato no tempo em que houve a alteração de escolaridade ou estado civil de um candidato, ou se houve mais de uma troca de partido no período entre eleições.

É importante destacar que este modelo partiu de uma análise empírica dos dados modelados e não é, de maneira alguma, definitivo. Experimentos realizados sobre modelos iniciais motivaram algumas alterações, enquanto desmotivaram outras. Experimentações exaustivas podem, potencialmente, a levar um esquema mais eficiente ou mais representativo. A seguir, algumas alternativas que poderiam ser consideradas:

- Nodos `Candidatura` poderiam se conectar diretamente aos nodos `Estado`, facilitando agregações e filtros por Estado. Esta alternativa não foi adotada para evitar o problema com nodos densamente conectados, pois um relacionamento a mais para cada `Candidatura` poderia degradar consultas que não filtrassem por estado

- Nodos `DadosTemporais` poderiam não existir, eliminando um relacionamento o nodo `Candidatura`. Esta alternativa não foi adotada para manter a simplicidade do esquema e evitar a criação de índices para relacionamentos.

- Outros relacionamentos fortemente tipados, como o que existe entre `Eleições` e `Ano` poderiam existir, mas sua experimentação mostrou resultados inconsistentes e os relacionamentos descritos acima foram mantidos.

- Outras estruturas para indexação das candidaturas poderiam ser experimentadas, como por exemplo indexar candidaturas por `Ano` e `Estado`. A discussão dos resultados tratará deste caso, ao abordar uma consulta em específico.

4.4 Considerações Finais

O modelo temporal para bancos de dados de grafos proposto por Cattuto et al. (2013) pode ser adaptado para este trabalho. Apesar de simples, as informações a respeito das candidaturas para vereador ao longo dos anos apresentam a característica temporal desejada para este trabalho. Além disso, como mostrou a subseção que trata de ruídos, ela muito se assemelha com o tipo de informação encontrada na web, com ruídos e discrepâncias e necessidade de pré-processamento, o que está alinhado com a discussão apresentada no Capítulo 3 acerca das motivações para o uso de bancos de dados de grafos.

O diferencial do presente trabalho com relação aos demais trabalhos citados na seção 3.1 a utilização de um modelo baseado em grafo implementado em um SGBD que armazena e processa osdados nativamente como um grafo. Isso permite uma modelagem mais natural dos dados, pois a estrutura de grafos permite melhor visibilidade ao usuário; consultas podem fazer referência diretamente à estrutura do grafo, com alto grau de abstração, possuindo funções nativas para consultas do tipo menor caminho ou a subgrafos; e as consultas são localizadas para uma parte do grafo o que não degrada muito o desempenho de consultas que envolvem relacionamentos.

A base de dados resultante da análise dos dados, pré-processamento e criação do banco de dados em Neo4j, apresentada neste capítulo, é relacionável com o trabalho de Cattuto et al. (2013), referência para a modelagem temporal. Assim, apresenta-se no próximo capítulo a experimentação realizada sobre o banco de dados construído, com uma análise dos resultados, discutindo as vantagens e os problemas de todas as decisões apresentadas.

5 EXPERIMENTOS

O capítulo anterior descreveu a construção de um modelo de dados que contempla as informações de eleições para vereadores em suas três eleições passadas. Como forma de avaliar esse modelo, este capítulo descreve os experimentos realizados sobre esta base de dados, considerando não somente o desempenho, em tempo, para diversas consultas, como também a obtenção de resultados semanticamente relevantes.

Para isso, é importantes considerar características de software e hardware do ambiente de execução desses experimentos, bem como o tamanho da base de dados, em número de nodos e relacionamentos. Estas características, as consultas realizadas ao banco, juntamente com a sua semântica, a metodologia de medição de desempenho destas consultas e a discussão a respeito dos resultados obtidos são apresentadas a seguir.

5.1 Ambiente de Execução

A criação da base de dados e os experimentos descritos neste trabalho foram realizados em um computador pessoal com as características apresentadas a seguir. É preciso considerar que, por se tratar de um computador pessoal, pode-se obter reduções nos tempos de execução ao se executar os mesmos procedimentos em um ambiente preparado, como um servidor, com o software limitado ao necessário para o experimento. De qualquer maneira, os resultados obtidos atendem aos requisitos deste trabalho. Considerando o hardware, o computador possui, em linhas gerais, as seguintes características:

- **Pocessador:** *Pentium(R) Dual-Core CPU E5400 @ 2.70GHz (2C 2.7GHz, 2MB L2)*
- **Memória:** dois módulos, somando 3GiB (sem dual-channel): *2GB DIMM DDR2 PC2-6400U DDR2-800* e *1GB DIMM DDR2 PC2-6400U DDR2-800*
- **Disco:** *160GB, SATA150, 3.5", 7200rpm, 8MB Cache*

Quanto ao software, considera-se relevante mencionar:

- **Sistema Operacional:** *Microsoft Windows 8.1 Professional 6.03.9600 x64*

- **Java Runtime Engine:** *JRE 1.7.0*
- **SGBD:** *Neo4j 2.1.1*

Também é importante citar as configurações de execução do Neo4j:

- **Java Heap e Garbage Collector:** de acordo com as recomendações da referência do Neo4j[1], foram definidas no arquivo `neo4j-server.properties` a configuração do tamanho máximo de memória ocupado pelo Java para executar o SGBD (1,8GiB, dos 3GiB disponíveis), com o parâmetro `-Xmx1800M`, e o algoritmo do *garbage collector*, com o parâmetro `XX:+UseConcMarkSweepGC`.
- **Memory Mapping:** conforme recomendação do Neo4j[2], o uso de mapeamento de arquivos em memória foi ativado no arquivo `neo4j.properties` com valores proporcionais ao total de memória disponível:

```
neostore.nodestore.db.mapped_memory = 90M
neostore.relationshipstore.db.mapped_memory = 800M
neostore.propertystore.db.mapped_memory=300M
neostore.propertystore.db.strings.mapped_memory = 300M
neostore.propertystore.db.arrays.mapped_memory = 10M
```

Essas configurações não foram estressadas, ou seja, seguiu-se apenas a configuração recomendada pelo manual do Neo4j, em seu guia de configurações e performance[3], sem procurar combinações que atingissem o melhor desempenho. Sugere-se pesquisar sobre as diversas opções de configuração, de forma a obter a performance máxima possível no ambiente de execução disponível, mas este não é o foco deste trabalho.

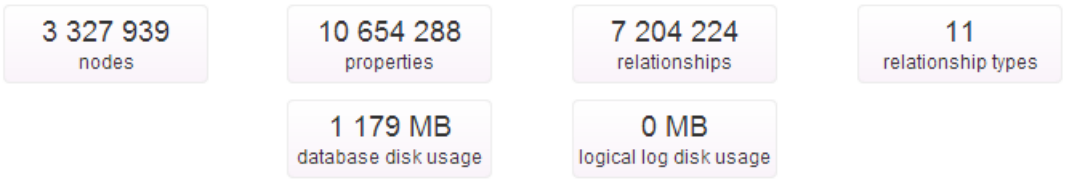
Após essas configurações e inicialização da base de dados criada conforme descrito no Capítulo 4, o ambiente está pronto para execução de consultas. A Figura 5.1 mostra as informações agregadas do banco de dados criado, disponibilizadas pelo SGDB, que indica uma base com mais de 3 milhões de nodos e 7 milhões de relacionamentos, ocupando mais de 1GiB em disco.

Para melhor compreensão das consultas e seus tempos de execução, cabe considerar também o número de nodos de cada tipo (`label`):

- O nodo `Eleicoes` é unitário, por ser o nodo de referência; há um nodo para cada ano considerado e, portanto, são 3 nodos `Ano`.
- Existem 5.666 nodos `Cidade` e 26 nodos `Estado`.

[1] http://docs.neo4j.org/chunked/stable/configuration-jvm.html
[2] http://docs.neo4j.org/chunked/stable/configuration-io-examples.html
[3] http://docs.neo4j.org/chunked/stable/configuration.html

Figura 5.1: Informações gerais sobre a base de dados, disponibilizadas pelo WebAdmin do Neo4j



Fonte: Elaborado pelo autor.

- São 37 nodos `Partido` e 68.794 nodos `Coligacao`.
- Ao todo, há 1.171.352 nodos `Candidatura`; há o mesmo número de nodos com o `label DadosTemporais`.
- Nodos `Candidato` somam 909.463.

Com este volume de dados, a criação de índices se fez necessária, pois consultas iniciais sem índices executaram em tempos impraticáveis. A seguir, as primitivas Cypher para os índices criados:

```
CREATE INDEX ON: Ano(ano)
CREATE INDEX ON: Estado(sigla)
CREATE INDEX ON: Cidade(nome)
CREATE INDEX ON: Partido(sigla)
CREATE INDEX ON: Candidatura(eleito)
CREATE INDEX ON: Candidatura(despesa)
CREATE INDEX ON: Candidatura(nome)
CREATE INDEX ON: Candidato(nome)
CREATE INDEX ON: Candidato(sexo)
```

5.2 Metodologia dos Experimentos

Os experimentos realizados neste trabalho visaram aferir o tempo médio de execução de consultas de diversos tipos. Estas medições precisam ser sistemáticas e padronizadas, de maneira a não apresentar resultados pouco representativos ou viciados. A metodologia utilizada para cada consulta foi a seguinte:

- Cada consulta foi executada 40 vezes, armazenando-se o tempo de execução para cada uma delas. Excluiu-se, então, os 4 piores e 4 melhores tempos e calculou-se a média dos valores remanescentes. Este valor foi considerado como o "Valor

médio 10% - 90%"das consultas, uma vez que os primeiros e últimos 10% foram desconsiderados, o que é importante pois esta média não deve considerar quaisquer desvios (como por exemplo a primeira execução de uma consulta, que não conta com o benefício da cache).

- Como os primeiros e últimos valores foram excluídos do valor de média, armazenou-se o menor e o maior tempos entre estas 40 execuções, para que fosse possível visualizar o menor e o pior casos.

- Para analisar o impacto da cache, uma segunda etapa de 5 execuções foi executada, desta vez indicando para o SGBD as opções `cache_type = none`, `use_memory_mapped_buffers = false`, `query_cache_size = 1`, que desativam o uso de cache e *memory mapping*, bem como o armazenamento de planos de execução para consultas. Isso força a execução de uma consulta em sua totalidade, sem que seja possível o aproveitamento de qualquer processamento prévio, o que permite visualizar o cenário de pior caso em uma situação ainda mais extrema.

- Por fim, calculou-se o menor tempo entre as execuções da primeira etapa e o maior tempo entre todas as execuções, ou seja, o maior valor entre as 40 consultas com a configuração padrão de execução e as 5 consultas sem cache (onde se observou a maioria dos tempos de pior caso).

As repetições foram executadas com o auxílio de um script Python, usando as funcionalidades do Rest Client, apresentado na Seção 4.2.2. Pode-se considerar um pequeno *overhead* de tempo para requisições enviadas ao servidor, pois a totalização foi feita através da marcação de tempo, no Python, antes e depois do envio da requisição. Esses tempos não causaram um impacto importante: durante testes, execuções usando este método mostraram tempos semelhantes aos exibidos pelo shell do Neo4j para a mesma consulta. Deve-se salientar que, se as mesmas consultas forem executadas pela interface web disponibilizada pelo Neo4j, que apresenta os resultados em forma de grafo (quando aplicável) ou listas estilizadas pela interface do navegador, elas apresentarão tempos superiores (assumindo que incluem o *overhead* de apresentação).

A próxima seção apresenta os resultados para diversas consultas, considerando a metodologia descrita acima.

5.3 Consultas Executadas

Para avaliar o modelo de dados e justificar sua correção e utilidade, considerou-se dois aspectos: a execução de consultas em tempo aceitável e a possibilidade de extrair informações relevantes. Para efetivar essa avaliação, este trabalho considerou diversos

tipos de consultas, que foram medidas em tempo de execução para sua avaliação. Além disso, diversas outras consultas, apesar de complexas e mais lentas, representam consultas menos corriqueiras e que fazem observações estatísticas a respeito dos dados.

Para todas estas consultas, utilizou-se a linguagem Cypher. A documentação do Neo4j disponibiliza o *Cypher Refcard 2.0*[4], que foi constantemente acessado durante o desenvolvimento das consultas e que é recomendado como auxílio para o acompanhamento do leitor.

Assim, dividiu-se as consultas em vários grupos, de acordo com características em comum entre elas. A seguir, apresenta-se estas consultas, com uma numeração sequencial e descrição de seu significado. Para aquelas consideradas mais representativas, são exibidos também os resultados obtidos, em forma de grafo ou de lista. Para cada subgrupo, então, tem-se agrupadas as consultas e os respectivos tempos de execução, conforme a metodologia descrita na Seção 5.2.

5.3.1 Consultas Não Temporais

O modelo criado para este trabalho considerou a necessidade de consultas temporais aos dados. Porém, nem todas as consultas são necessariamente temporais. Considera-se como consultas não temporais todas aquelas que não fazem nenhum filtro por atributos temporais. É importante analisá-las para confirmar que o impacto da indexação temporal presente no modelo é mínimo para as consultas que não se utilizam desta propriedade.

Consulta 1. Acessar todos os dados (os nodos relacionados) a um determinado candidato, a partir de seu `ID`, usando busca em profundidade:

```
MATCH (m)-[*0..2]->(ca:Candidatura)->cand,
  ca-[*..]->(f)
WHERE id(cand) = 2817217
RETURN cand,m,ca,f
```

O resultado desta consulta é o grafo que aparece na Figura 5.2, que mostra todos os nodos e relacionamentos relacionados ao candidato selecionado (destacado em amarelo).

Consulta 2. Consultar por um nome específico (retornando os nodos dos registros encontrados):

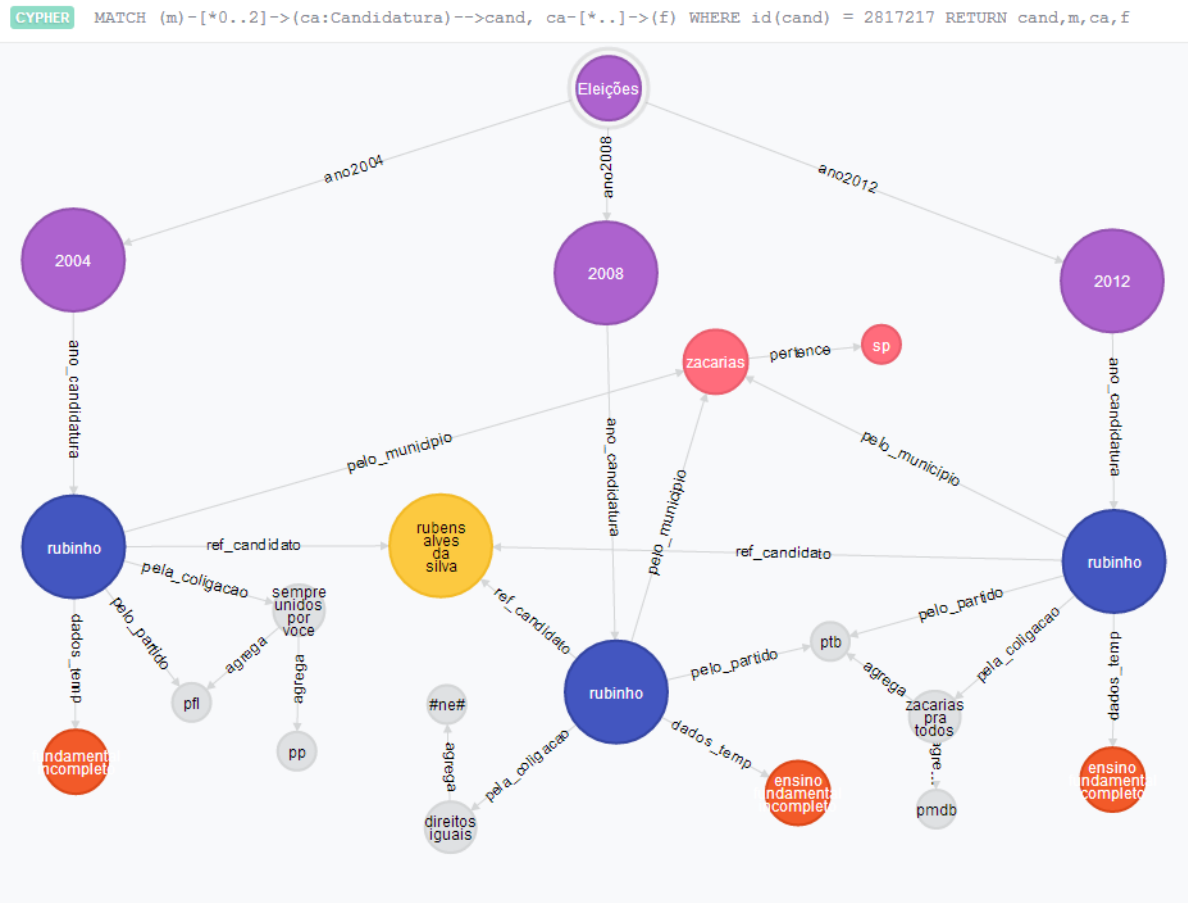
```
MATCH (cand:Candidato)
WHERE cand.nome = 'joao da silva'
RETURN  cand
```

Consulta 3. Consultar por nomes candidatos usando expressão regular, (retornando apenas os nomes como resultado):

```
MATCH (cand:Candidato)
```

[4] http://docs.neo4j.org/refcard/2.0/

Figura 5.2: Grafo com todos os nodos relacionados diretamente a um candidato específico (Consulta 1).



Fonte: Elaborado pelo autor.

```
WHERE cand.nome =~ '.*ceratti.*'
RETURN cand.nome AS nome_candidato
```

Consulta 4. Buscar os vereadores eleitos pela mesma coligação do candidato `c`, selecionado por `ID`:

```
MATCH (c:Candidato)<-[:ref_candidato]-(ca:Candidatura)
    -[:pela_coligacao]->(co:Coligacao)<-[:pela_coligacao]
    -(ca2:Candidatura)-[:ref_candidato]->(c2:Candidato),
  (a:Ano)-[:ano_candidatura]->(ca),(ca)-[:pelo_partido]
  ->(p:Partido), (co)-[:agrega]->(pco:Partido)
WHERE id(c) = 2734551 AND ca2.eleito = 1
RETURN  a.ano AS ano,c.nome+ (+p.sigla+) AS candidato,
  collect(distinct c2.nome) AS cands_mesma_coligacao,
  collect(distinct pco.sigla) AS partidos_da_coligacao
```

```
ORDER BY ano ASC
```

Apesar de o resultado incluir o ano de cada candidatura do candidato `c`, como mostra a Figura 5.3, a consulta não acessa o índice temporal e, por isso, é considerada não-temporal.

Figura 5.3: Candidatos eleitos pela mesma coligação pela qual concorreu o candidato `c` (Consulta 4).

ano	candidato	cands_mesma_coligacao
2004	yusif ali chahin (pps)	william boss woo, wadih jorge mutran, ushitaro kamia, miriam athie, marcos antonio zerbini, lenice lemos sao bernardo, juscelino jose ataliba antonio gadelha, jose ricardo franco montoro, jose police neto, jose anibal peres de pontes, jose roberto nazello de alvarenga tripoli, jorge luis borges, gilson almeida barreto, gilberto tanos natalini, edivaldo alves estima, domingos odone dissei, dalton silvano do amaral, claudio do prado nogueira, carlos alberto eugenio apolinario, carlos alberto de quadros bezerra junior, aurelio nomura, aguinaldo timotheo pereira, ademir da guia
2008	yusif ali chahin (psdb)	ricardo teixeira, mara cristina gabrilli, juscelino jose ataliba antonio gadelha, jose police neto, jose souza dos santos, gilson almeida barreto, gabriel benedito issaac chalita, claudio roberto barbosa de souza, carlos alberto de quadros bezerra junior, antonio floriano pereira pesaro, adolfo quintas goncalves neto
2012	yusif ali chahin (pps)	ari friedenbach, ricardo young silva

Fonte: Elaborado pelo autor.

A Tabela 5.1 mostra os tempos de execução obtidos para cada consulta.

Tabela 5.1: Tempos de execução para consultas não-temporais (em ms)

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 1	89	78 - 172
Consulta 2	17	16 - 125
Consulta 3	3.006	2.953 - 54.688
Consulta 4	16	0[5] - 312

Fonte: Elaborado pelo autor.

Os tempos de execução levantados para as consultas não temporais mostram um dado importante: a Consulta 1 permite, para um dado candidato, obter todas as informações referentes a ele em um tempo médio abaixo de 100ms. Além disso, esta consulta merece destaque por usar um artifício da estrutura de grafos: a busca por profundidade. Além de simples e legível, a *query* Cypher traz um resultado que expressa visualmente

[5]O tempo 0 deve ser interpretado apenas como "muito próximo de zero". O valor se refere à diferença entre os tempos anterior e posterior à execução da consulta pelo script Python e pode apresentar pequenos desvios de precisão.

a estrutura do grafo e pode ser diretamente associada ao esquema apresentado na Figura 4.6. A mesma consulta em um modelo relacional seria um acumulado de `JOINs` que, além de demandar um maior conhecimento prévio do esquema de dados, resultaria em um código pouco legível. Apesar de somente a experimentação, com um modelo temporal semelhante para banco de dados relacional, poder comparar os tempos de execução, confirma-se aqui algumas das vantagens citadas na Seção 3.2 como motivação para o uso de SGBD de grafos.

A Consulta 2 mostra a pesquisa por um nome exato de candidato que, apesar de rápida, não expressa uma necessidade comum de usuários buscando em uma base de dados. Esta necessidade seria melhor atendida pela Consulta 3, que pesquisa os nomes que contém uma determinada *substring*. Esta consulta teria a mesma utilidade de um padrão *%like%* em SQL. Infelizmente, o tempo de execução obtido aqui é impraticável, principalmente se considerarmos o pior caso, onde passa de 50 segundos. Obviamente, se faz necessária a busca de alternativas para a execução de consultas deste tipo.

De maneira geral, considerando também a Consulta 4, percebe-se que o índice temporal não afeta diretamente os tempos de execução para consultas não-temporais, que podem ser feitas simplesmente sem acessar o índice temporal.

5.3.2 Consultas Por Instante no Tempo

Consultas temporais devem permitir a visualização do estado dos dados em um instante no tempo. No caso da base de dados deste trabalho, que possui granularidade temporal baixa, os pontos específicos no tempo a considerar são apenas os anos das eleições consideradas. Assim, pode-se filtrar pelo ano desejado explicitando-o na consulta, ou pré processar o índice temporal e encontrar o valor correspondente ao ano mais recente existente. A seguir, exemplifica-se como são feitas essas consultas.

Consulta 5. Contabilizar o número de vereadores (candidatos eleitos) de um município em um ano específico (2008).

```
MATCH (e:Eleicoes)-[:ano2008]->(a:Ano)-[:ano_candidatura]->
  (ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(est:Estado)
WHERE est.sigla = 'sp' AND c.nome = 'sao paulo'
  AND ca.eleito = 1
RETURN count(ca) AS vereadores_2008,c.nome AS cidade
```

Consulta 6. Listar o número total de candidatos de um estado, separados por cidade, no ano mais recente (o da última eleição registrada).

```
MATCH (a:Ano)
WITH  max(a.ano) AS mais_recente
MATCH (e:Eleicoes)->(a:Ano)-[:ano_candidatura]
```

```
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(est:Estado)
WHERE est.sigla = 'sp' AND a.ano = mais_recente
RETURN  count(ca) AS candidaturas_por_cidade,
  sum(ca.eleito) AS eleitos,c.nome AS cidade
ORDER BY candidaturas_por_cidade desc
```

Consulta 7. Obter a despesa média por campanha para cada partido na última eleição registrada.

```
MATCH (a:Ano)
WITH max(a.ano) AS mais_recente
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]->
  (ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(e:Estado),
  ca-[:pelo_partido]->(p:Partido)
WHERE e.sigla = 'rs' AND c.nome = 'porto alegre'
  AND a.ano = mais_recente AND ca.despesa > 0
RETURN round(avg(ca.despesa)*100)/100 AS
  despesas_por_partido,p.sigla
ORDER BY despesas_por_partido DESC
```

Consulta 8. Partidos que se coligaram com PX em todo o estado do RS, no ano de2012.

```
MATCH (el:Eleicoes)-[:ano2012]->(a:Ano)->
  (ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(e:Estado sigla:  'rs' ),
  (co)-[:agrega]->(part2:Partido)
  ->(part:Partido sigla:  'px'[6]),
  (co)-[:agrega]->(part2:Partido)
RETURN  part.sigla as partido_p,
  collect(distinct part2.sigla) as partidos_coligados_com_p
```

A Figura 5.4 mostra o resultado da Consulta 6, que exemplifica uma consulta por dados referentes a um determinado instante no tempo. O ano, portanto, não aparece nos resultados, já que o resultado está indexado por um ano específico (neste caso, o mais recente, que foi pré-calculado pela *query* Cypher). Essa mesma característica se observa nas demais consultas. Consultas cujo resultado envolve a exposição de dados agregados de um ou mais partidos políticos não terão seu resultado exibido neste trabalho.

[6]PX é um partido fictício utilizado para preservar informações sobre partidos políticos neste trabalho. A consulta foi executada usando a sigla de um partido existente.

Figura 5.4: Número total de candidatos de um estado, separados por cidade, na eleição mais recentemente registrada(Consulta 6).

Fonte: Elaborado pelo autor.

A partir da Tabela 5.2, que mostra os tempos de execução das consultas por instante no tempo (um ponto específico no índice temporal), é possível fazer inferências sobre a eficiência do modelo apresentado neste trabalho.

Tabela 5.2: Tempos de execução para consultas a um ponto específico no tempo (em ms)

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 5	17	16 - 312
Consulta 6	3.365	3.266 - 13.875
Consulta 7	931	906 - 5.407
Consulta 8	10.193	10.109 - 18.719

Fonte: Elaborado pelo autor.

O resultado para a Consulta 5, que seleciona os dados para um município específico, mostra que o modelo construído é eficiente para selecionar informações sobre eleições em um ano específico. O problema é que essa eficiência só vale para um subgrafo pequeno. Quando a agregação compreende um estado inteiro, ou seja, um subgrafo muito maior, como na Consulta 6, a performance se deteriora.

Uma das alternativas para este problema seria vincular o índice geográfico ao índice temporal. Assim, o estado SP, por exemplo, não teria apenas uma instância, e sim uma instância para cada eleição. Essa alternativa implicaria na indexação hierárquica das elei-

ções do estado SP no ano de 2012 da seguinte maneira:

```
MATCH (e:Eleicoes)-[:ano2012]->(a:Ano)-[:pelo_estado_SP]
  ->(est:Estado)-[:est_candidatura]->(ca:Candidatura)
```

Esta estrutura permitira o acesso a um determinado subgrafo usando a hierarquia multinível descrita como árvore de caminhamento (*path three*[7]). Esta solução tem a vantagem de evitar a consulta às propriedades de todos os nodos, usando apenas os relacionamentos fortemente tipados para filtrar o subgrafo desejado. Segundo Cattuto et al. (2013), os benefícios do uso desta hierarquia são uma questão em aberto, já que, em seu trabalho, não foi feita a comparação específica entre as alternativas.

Durante este trabalho, a estrutura hierárquica incluindo indexação geográfica foi experimentada. Consultas como as de número 6 e 8 apresentaram reduções consideráveis em seus tempos de execução. Entretanto, diversas consultas mais simples, que não fazem uso simultâneo dos índices temporal e geográfico, ficaram mais lentas. Por causa deste impacto, esta alternativa de modelagem não foi adotada. Não apresenta-se aqui os resultados desta comparação, pois ela não seguiu uma metodologia definida nem foi exaustiva, mas sugere-se esta noção de modelagem para experimentações futuras.

Uma segunda solução, também considerada por Cattuto et al. (2013), seria realizar o pré-processamento de consultas muito complexas. De fato, considerando o volume de dados tratados por este trabalho, poderia-se armazenar, por exemplo, estatísticas gerais para as eleições de cada estado em um determinado ano como um atributo de numa nova relação (Ano)–>(Estado), o que reduziria drasticamente o tempo para determinadas consultas. Para agregações mais simples, como as Consultas 5 e 7, o modelo atual é satisfatório.

5.3.3 Consultas Por Período

As consultas por período são aquelas que estabelecem mais de um ponto no tempo, seja especificando explicitamente cada um destes pontos, definindo um limite inferior e/ou superior ou especificando um intervalo dentro do qual os dados devem se encontrar. Na base de dados discutida neste trabalho, estas consultas obrigam a inserção de um filtro pelo valor `ano`, atributo dos nodos de *label* `Ano`.

Consulta 9. Número de candidatos em um município por período com limite inferior e superior definidos (2008 a 2012).

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado)
WHERE e.sigla = 'sp' AND c.nome='sao paulo'
  AND a.ano >= 2008 AND a.ano <= 2012
RETURN  count(ca) AS candidatos_no_ano, a.ano AS ano
```

[7] http://docs.neo4j.org/chunked/stable/cypher-cookbook-path-tree.html

```
ORDER BY ano ASC
```

Consulta 10. Número de candidatos em um município por período definido por diversos pontos no tempo (o período pode ter sido previamente tratado).

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado)
WHERE e.sigla = 'sp' AND c.nome='sao paulo'
  AND a.ano IN [2004,2008]
RETURN  count(ca) AS candidatos_no_ano,a.ano AS ano
ORDER BY ano ASC
```

Consulta 11. Número de candidatos em um município por período com limites inferior como a eleição mais antiga registrada e superior com a mais recente (pré-selecionados).

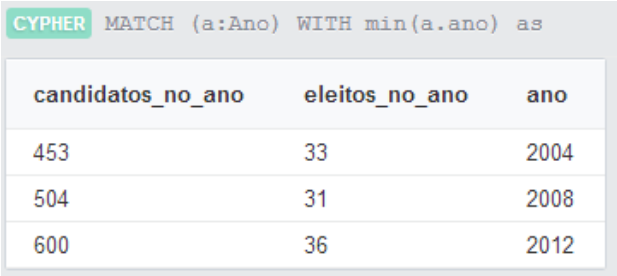
```
MATCH (a:Ano)
WITH  min(a.ano) AS mais_antigo,max(a.ano) AS mais_recente
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado)
WHERE e.sigla = 'rs' AND c.nome = 'porto alegre'
  AND a.ano >= mais_antigo AND a.ano <= mais_recente
RETURN  count(ca) AS candidatos_no_ano,
  sum(ca.eleito) AS eleitos_no_ano,a.ano AS ano
ORDER BY ano ASC
```

Consulta 12. Número de candidatos em um município por período contido em uma extensão determinada (*in range*).

```
MATCH (a:Ano)
WITH  min(a.ano) AS mais_antigo,max(a.ano) AS mais_recente
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado)
WHERE e.sigla = 'rs' AND c.nome = 'porto alegre'
  AND a.ano IN range(mais_antigo,mais_recente)
RETURN  count(ca) AS candidatos_no_ano,
  sum(ca.eleito) AS eleitos_no_ano,a.ano AS ano
ORDER BY ano ASC
```

Todas as consultas desta subseção apresentam resultados semelhantes e podem ser exemplificados pela Figura 5.5, que mostra os dados selecionados em um determinado período, alinhados aos anos associados.

Figura 5.5: Número de candidatos em um município por período de determinado (*in range*, Consulta 12).



Fonte: Elaborado pelo autor.

Tabela 5.3: Tempos de execução para consultas por um período no tempo (medidos em ms).

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 9	90	78 - 250
Consulta 10	95	94 - 266
Consulta 11	704	688 - 2.859
Consulta 12	708	688 - 2.922

Fonte: Elaborado pelo autor.

A Tabela 5.3 apresenta a eficiência do modelo de dados para consultas que abrangem faixas de tempo específicas. A análise dos tempos é semelhante à já feita anteriormente: em subgrafos pequenos, como o das candidaturas para uma cidade específica, as consultas têm tempo médio abaixo de 100ms. As consultas que fizeram uma pré-seleção dos anos mais antigo e mais próximo apresentaram uma degradação significativa na performance. As consultas 11 e 12, que selecionam pelo município Porto Alegre, abrangem um conjunto de dados menor que as consultas 9 e 10, que englobam as candidaturas para a cidade de São Paulo; mesmo assim, elas foram mais lentas, principalmente se considerados os piores tempos registrados. Apesar de essas consultas ficarem genéricas e poderem ser reusadas, do jeito como foram escritas, mesmo na inclusão de informações sobre novos anos, a penalidade na performance é muito alta. Deve-se experimentar o acesso ao nodo ano a partir do nodo de referência, pois a subpressão desta relação mostrou resultados piores em relação à seleção direta do ano, conforme feito aqui. Essa comparação aparece na comparaCypher. Outra alternativa é calcular previamente os limites inferior e superior (ano mais antigo e ano mais recente), enviando-os como parâmetro para a consulta.

É importante salientar que essas consultas apresentam diversas maneiras de se filtrar por um período do tempo: através da comparação com limites inferior e superior; através da definição de um intervalo (*range*); e através da seleção de todos os tempos válidos

para o intervalo (cláusula *IN*). Neste último caso, a consulta propriamente dita não filtra por um período de tempo, pois apenas seleciona os anos que estão inclusos em uma coleção definida; na prática, essa solução representaria um período se os valores enviados à cláusula *IN* tivessem sido pré-processadas, calculando-se previamente todos os pontos no tempo válidos para um dado intervalo. Essa mesma consulta poderia, ainda, ser feita usando relacionamentos fortemente tipados, pois a linguagem Cypher permite selecionar um relacionamento usando mais de um tipo como parâmetro. Entretanto, essa consulta implica em pré-processar o parâmetro ano e resulta em uma *query* menos legível:

```
MATCH (el:Eleicoes)-[:ano2008|ano2012]->(a:Ano)
```

Sugere-se a experimentação desta consulta ao incluir informações sobre outros anos na base de dados atual, ou então com o modelo reestruturado conforme o discutido na Subseção 5.3.3 (com indexação temporal e geográfica em hierarquia), caso em que possivelmente haverá o benefício dos relacionamentos fortemente tipados. Entretanto, as quatro consultas executadas são satisfatórias para o modelo deste trabalho, com tempos médios de execução abaixo de um segundo.

5.3.4 Consultas por Evolução no Tempo

As próximas consultas analisadas têm por objetivo elucidar a possibilidade de selecionar os dados em período de tempo. Entretanto, deve ser possível extrair informações relevantes sobre a evolução dos dados a partir do modelo, indicando padrões nas alterações ocorridas ao longo do tempo. As consultas apresentadas a seguir apresentam esta característica.

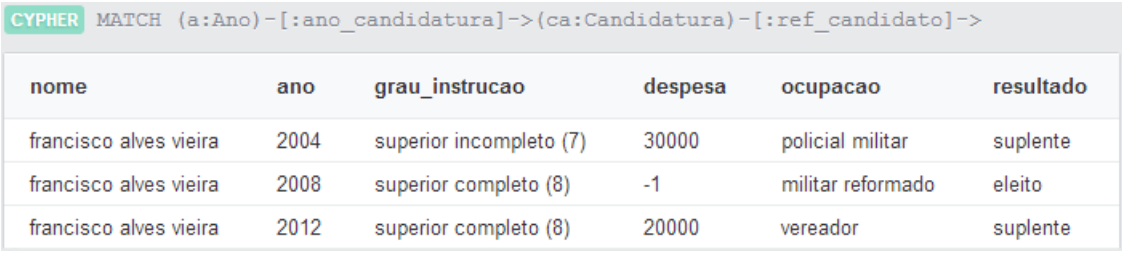
Consulta 13. Evolução das informações temporais de um candidato (selecionado por `ID` ao longo dos anos.

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:ref_candidato]->(c:Candidato),
  (ca)-[:dados_temp]->(t:DadosTemporais)
WHERE id(c) = 361
RETURN  c.nome AS nome,a.ano AS ano,t.grau_instrucao
  +' ('+str(t.cod_grau_instrucao)+')' AS grau_instrucao,
  ca.despesa AS despesa, t.ocupacao AS ocupacao,
  ca.resultado AS resultado
```

O resultado desta consulta, apresentado na Figura 5.6, é bastante representativo, pois mostra a evolução de diversos dados de um determinado candidato que participou de todas as eleições consideradas. A estrutura das demais consultas por evolução no tempo seguem o mesmo formato, com uma determinada informação extraída para cada ano.

Com essa consulta, é possível extrair informações importantes a respeito do candidato ao longo dos anos. Observa-se que, de 2004 para 2008, o candidato selecionado evoluiu

Figura 5.6: Evolução dos dados temporais de um candidato específico (selecionado por `ID`) ao longo dos anos.



```
CYPHER MATCH (a:Ano)-[:ano_candidatura]->(ca:Candidatura)-[:ref_candidato]->
```

nome	ano	grau_instrucao	despesa	ocupacao	resultado
francisco alves vieira	2004	superior incompleto (7)	30000	policial militar	suplente
francisco alves vieira	2008	superior completo (8)	-1	militar reformado	eleito
francisco alves vieira	2012	superior completo (8)	20000	vereador	suplente

Fonte: Elaborado pelo autor.

em escolaridade, completando o ensino superior, e foi movido para a reforma militar. No ano de 2008, o candidato foi eleito vereador, e esta é a sua profissão no ano de 2012, onde, em uma nova candidatura, o candidato ficou como suplente. Infelizmente a comparação de despesa não foi possível porque, conforme observou-se com esta e outras consultas, as candidaturas do ano 2008 não registraram a informação de despesa de campanha declarada (todos os valores, nos arquivos CSV, aparecem como -1).

Consulta 14. Evolução da despesas de campanha em um determinado município, para o partido `p` (selecionado por `sigla`).

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado),
  ca-[:pelo_partido]->(p:Partido)
WHERE e.sigla = 'rs' AND c.nome = 'porto alegre'
  AND p.sigla = 'px'[8]
RETURN a.ano AS ano,p.sigla AS partido,
  sum(ca.despesa) AS despesa_total
ORDER BY a.ano ASC
```

Consulta 15. Evolução do número de vereadores eleitos em um determinado município, para o partido `p` (selecionado por `sigla`).

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado),
  ca-[:pelo_partido]->(p:Partido)
WHERE e.sigla = 'rs' AND c.nome = 'porto alegre'
AND p.sigla = 'px'[8] AND ca.eleito = 1
```

[8]PX é um partido fictício utilizado para preservar informações sobre partidos políticos neste trabalho. A consulta foi executada usando a sigla de um partido existente.

```
RETURN a.ano AS ano,p.sigla AS partido,
  sum(ca.eleito) AS eleitos_pelo_partido
ORDER BY a.ano ASC
```

Consulta 16. Evolução do número total de candidaturas ao longo dos anos.

```
MATCH (a:Ano)->(ca:Candidatura)
RETURN a.ano,count(ca)
ORDER BY a.ano ASC
```

As Consultas 14 e 15 são muito parecidas, embora a segunda seja um pouco mais complicada por envolver agregação. A Consulta 16, entretanto, é presumivelmente muito lenta, pois abrange a totalidade de nodos com o label `Candidatura`. A Tabela 5.4 mostra os tempos de execução para estas consultas, além da Consulta 13.

Tabela 5.4: Tempos de execução para consultas sobre evolução dos dados ao longo do tempo.

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 13	8	0[9] - 125
Consulta 14	80	78 - 219
Consulta 15	16	16 - 94
Consulta 16	10.791	10.656 - 35.516

Fonte: Elaborado pelo autor.

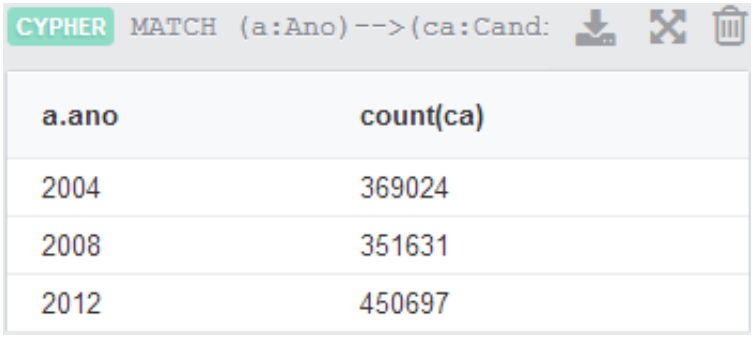
A Consulta 16, conforme esperado, é extremamente lenta. Ela é interessante, entretanto, porque revela uma informação importante a respeito do cenário eleitoral brasileiro no período considerado. A Figura 5.7, que mostra o resultado desta consulta, revela um dado surpreendente. Ao contrário do que poderia-se imaginar, o número de candidaturas caiu no ano de 2008, em relação ao ano de 2004. Esse número aumentou novamente em 2012. Esperava-se que o total de candidaturas fosse sempre crescente, mas uma breve pesquisa na web revela que alterações na legislação diminuíram o número de vagas para o ano 2008 e o aumentaram novamente para as eleições de 2012[10]. Pela relevância dos valores retornados por esta consulta, uma solução para o problema de performance seria, conforme já sugerido para outros casos, o pré-processamento do resultado, que poderia ser armazenado no nodo do ano correspondente.

As demais consultas (13, 14 e15) apresentaram tempos satisfatórios, com valores médios abaixo de 100ms. Mesmo considerando o pior caso, os tempos ficaram abaixo de

[9] O tempo 0 deve ser interpretado apenas como "muito próximo de zero". O valor se refere à diferença entre os tempos anterior e posterior à execução da consulta pelo script Python e pode apresentar pequenos desvios de precisão.

[10] http://www.tre-sc.jus.br/site/resenha-eleitoral/edicoes-impressas/integra/arquivo/2012/junho/artigos/numero-de-vagas-nas-camaras-municipais/

Figura 5.7: Evolução dos número total de candidaturas ao longo do tempo.



a.ano	count(ca)
2004	369024
2008	351631
2012	450697

Fonte: Elaborado pelo autor.

300ms, o que mostra uma eficiência importante a se considerar. Esse resultado mostra que é possível analisar a evolução dos dados temporais temporais de um candidato com uma consulta rápida. O mesmo vale para os dados agregados por um partido, que pode analisar o seu desempenho em um município e relacioná-lo, por exemplo, com os investimentos em campanha. São análises que permitem traçar um desenho eleitoral ao longo do tempo.

A complementação desta base de dados com outras informações pode trazer resultados ainda mais relevantes. Com a inserção de outros anos, é possível acompanhar o crescimento de um determinado partido político em um município ou região. Dados sobre a votação, com o total de votos para cada candidato, podem ajudar neste tipo de levantamento. O mesmo vale para candidatos, isoladamente. É possível combinar informações para discutir a influência de alguns fatores, como o investimento em campanha e o grau de escolaridade do candidato, na eleição de um candidato. Recentemente surgiu também a obrigatoriedade de prestação de contas, com a declaração do patrimônio de cada candidato; estes dados podem permitir fazer associações do poder aquisitivo de um candidato com o seu desempenho nas eleições.

As consultas sobre evolução dos dados integram possivelmente o conjunto mais importante entre todas as analisadas, pois permite análises complexas do cenário eleitoral e de sua evolução (com desempenho plenamente satisfatório).

5.3.5 Outras Consultas

A análise de consultas sobre evolução dos dados não só trouxe informações importantes sobre as eleições para vereadores no Brasil, como motivou a construção de consultas ainda mais expressivas. Por isso, construiu-se algumas consultas mais complexas que ajudam a justificar a relevância do modelo criado, justamente por trazerem descobertas que não seriam possíveis sem uma modelagem temporal.

Essas consultas, que mostram algumas alterações importantes no perfil dos candidatos

e partidos políticos, estão apresentadas a seguir.

Consulta 17. Evolução da participação feminina nas eleições para vereador no Rio Grande do Sul.

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]-(e:Estado),
  (ca)-[:ref_candidato]->(cn:Candidato)
WHERE e.sigla = 'rs'
WITH  sum(CASE WHEN cn.sexo = 'feminino' THEN 1 ELSE 0 END)
  AS mulheres_ano,count(ca) AS cand_ano,a.ano AS ano,
  sum(CASE WHEN cn.sexo = 'feminino' AND ca.eleito = 1
  THEN 1 ELSE 0 END)
  AS m_eleitas_ano, sum(CASE WHEN ca.eleito = 1
  THEN 1 ELSE 0 END)
  AS cand_eleitos_ano
RETURN  ano, mulheres_ano,cand_ano,
  round((toFloat(mulheres_ano)/toFloat(cand_ano)
  *10000))/100 AS perc_mulheres_ano,
  round((toFloat(m_eleitas_ano)/toFloat(cand_eleitos_ano)
*10000))/100 AS perc_eleitas_ano
ORDER BY ano ASC
```

Figura 5.8: Participação feminina nas eleições para vereador no RS.

CYPHER `MATCH (el:Eleicoes)-->(a:Ano)-[:ano_candidatura]->`

ano	mulheres_ano	cand_ano	perc_mulheres_ano	perc_eleitas_ano
2004	4847	22384	21.65	11.61
2008	4526	21078	21.47	11.91
2012	8885	26611	33.39	14.15

Fonte: Elaborado pelo autor.

A Consulta 17 é bastante complexa, mas seu resultado, exibido na figura 5.8, permite uma análise panorâmica a respeito da participação feminina nas eleições para vereador em um estado específico, apresentando não só a proporção de mulheres sobre o total de candidatos, como a mesma proporção considerando somente os vereadores eleitos. Assim, a partir desta consulta, pode-se concluir que recentemente houve um aumento significativo na participação das mulheres para o pleito de vereador. Apesar disso, o mesmo aumento

não foi observado na proporção entre os eleitos, ou seja, apesar de mais mulheres terem se candidato nas eleições de 2012, a proporção das que se elegeram vereadoras sobre o total não cresceu na mesma medida. A desproporção entre a participação feminina nas eleições e a participação feminina na câmara de vereadores é histórica[11].

Consulta 18. Trocas de partidos por candidatos de uma eleição para outra, classificadas pelo número de ocorrências.

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:ref_candidato]->(c:Candidato)
  <-[:ref_candidato]-(ca2:Candidatura)<-(a2:Ano),
  (ca)-[:pelo_municipio]->(cid:Cidade)-[:pertence]
  ->(est:Estado),
  (ca)-[:pelo_partido]->(p1:Partido),
  (ca2)-[:pelo_partido]->(p2:Partido)
WHERE est.sigla = 'rs' AND a2.ano = a.ano + 4 AND p1 <> p2
WITH p1.sigla+' -> '+p2.sigla AS troca,str(a.ano)+'->'
  +str(a2.ano) AS ano_troca,
  p1.numero+' -> '+p2.numero AS n_troca
RETURN  ano_troca,troca,n_troca,
  count(troca) AS total_trocas
ORDER BY total_trocas desc
```

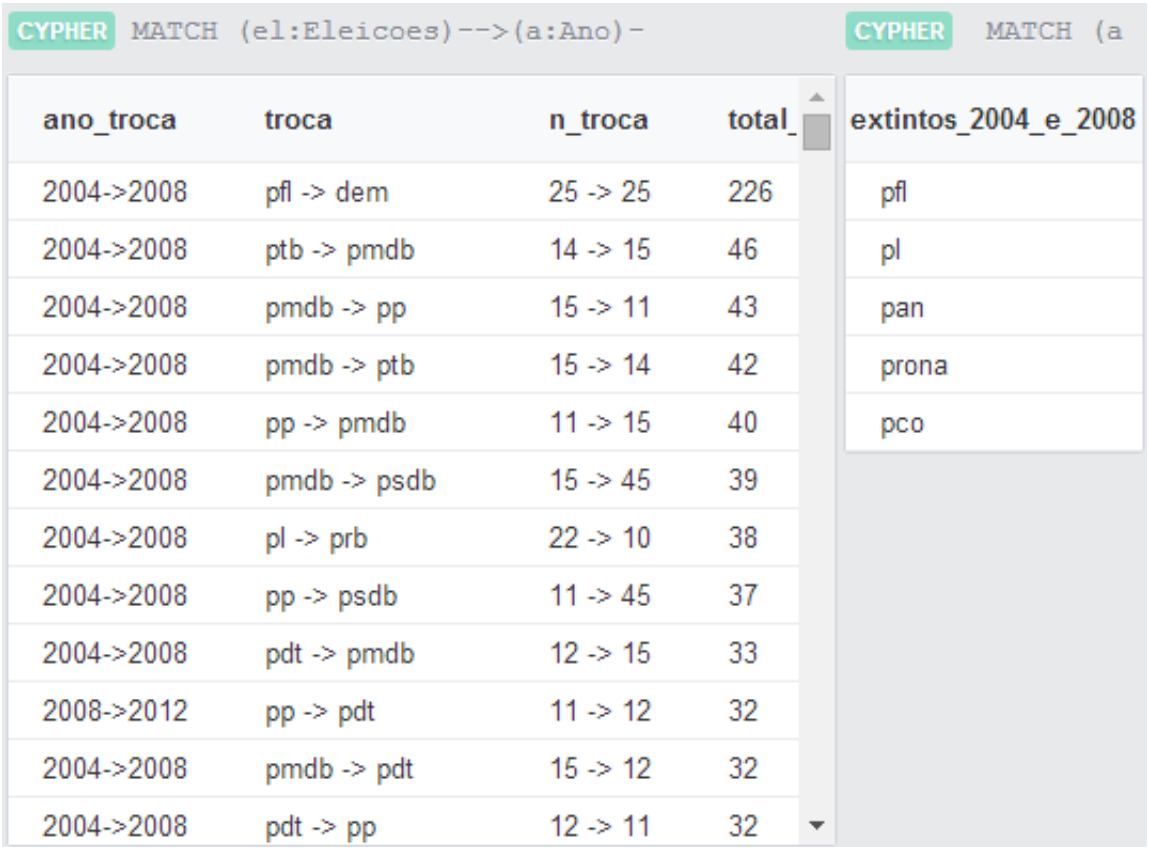
Consulta 19. Partidos extintos após as eleições de 2004 (considerando os que não apareceram em 2008 nas eleições do RS).

```
MATCH (a:Ano)-[:ano_candidatura]->(ca:Candidatura)
  -[:pelo_partido]->(p1:Partido),
  (ca)-[:pelo_municipio]->(cid:Cidade)
  -[:pertence]->(est:Estado)
WHERE est.sigla= 'rs' ANDa.ano=2008
WITH  collect(distinct p1.sigla) AS partidos_2008
MATCH (a2:Ano)-[:ano_candidatura]->(ca2:Candidatura)
  -[:pelo_partido]->(p2:Partido),
  (ca2)-[:pelo_municipio]->(cid2:Cidade)
  -[:pertence]->(est2:Estado)
WHERE est2.sigla= 'rs' AND a2.ano=2004 AND
  NOT (p2.sigla in (partidos_2008))
RETURN distinct p2.sigla AS extintos_entre_2004_e_2008
```

As Consultas 18 e 19 são análises a respeito da participação dos partidos em cada elei-

[11] http://www.xn–portaldaorganizao-nqb9e.org.br/wp-content/uploads/2012/08/102109190-Eleicoes-Municipais-de-2012-e-o-aumento-das-mulheres-nas-camaras-de-vereadores.pdf

Figura 5.9: Transições entre partidos para eleições subsequentes (à esquerda) e partidos que desapareceram em eleições subsequentes (à direita).



CYPHER MATCH (el:Eleicoes)-->(a:Ano)-

ano_troca	troca	n_troca	total_
2004->2008	pfl -> dem	25 -> 25	226
2004->2008	ptb -> pmdb	14 -> 15	46
2004->2008	pmdb -> pp	15 -> 11	43
2004->2008	pmdb -> ptb	15 -> 14	42
2004->2008	pp -> pmdb	11 -> 15	40
2004->2008	pmdb -> psdb	15 -> 45	39
2004->2008	pl -> prb	22 -> 10	38
2004->2008	pp -> psdb	11 -> 45	37
2004->2008	pdt -> pmdb	12 -> 15	33
2008->2012	pp -> pdt	11 -> 12	32
2004->2008	pmdb -> pdt	15 -> 12	32
2004->2008	pdt -> pp	12 -> 11	32

CYPHER MATCH (a

extintos_2004_e_2008
pfl
pl
pan
prona
pco

Fonte: Elaborado pelo autor.

ção e o resultado de ambas aparece na Figura 5.9. A primeira mostra as transições mais comuns de um partido para o outro: para duas eleições subsequentes, o par PX e PY e o número de candidatos fez essa troca de filiação. Além de revelar informações relevantes para os partidos políticos, esta consulta constatou um dado a respeito dos partidos: alguns partidos políticos mudaram de nome, apesar de serem ainda considerados o mesmo partido. A primeira linha do resultado da Consulta 18 mostra o expressivo número de trocas do partido PFL para o partido DEM. Na verdade, trata-se da mesma legenda, que apenas trocou de nome e sigla.

Esta descoberta foi o que motivou, a criação da Consulta 19. Através de uma análise considerando o estado do Rio Grande do Sul, foi possível constatar quais partidos foram extintos. Através de uma consulta rápida à web[12], foi possível confirmar que os partidos listados pela consulta foram, de fato, extintos, ao se fundirem com outros e criarem novos partidos (PL e PRONA, criando o PR), se incorporarem a partidos maiores (PAN, incorporado ao PR) ou simplesmente mudarem de nome (PFL, que virou DEM).

[12]http://pt.wikipedia.org/wiki/Anexo:Lista_de_partidos_pol%C3%ADticos_brasileiros_extintos

Consulta 20. Comparação entre melhoras no nível de escolaridade de candidatos eleitos/não-eleitos.

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:ref_candidato]->(c:Candidato)
  <-[:ref_candidato]
  -(ca2:Candidatura)<-(a2:Ano),
  (ca)-[:pelo_municipio]->(cid:Cidade)-[:pertence]
  ->(est:Estado),
  (ca)-[:dados_temp]->(t:DadosTemporais),
  (ca2)-[:dados_temp]->(t2:DadosTemporais)
WHERE est.sigla = 'rs' AND a2.ano = a.ano + 4
WITH CASE
  WHEN t2.cod_grau_instrucao = t.cod_grau_instrucao + 1
  THEN 1
  WHEN t2.cod_grau_instrucao > t.cod_grau_instrucao + 1
  THEN 2 ELSE 0 END AS melhora,
  str(a.ano)+'->'+str(a2.ano) AS ano_troca,
  ca.eleito AS eleito
RETURN  ano_troca,
  round(toFloat(SUM(CASE WHEN eleito = 0 AND melhora = 0
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 0
  THEN 1 else 0 END))*10000)/100 AS nao_eleitos_mantiveram,
  round(toFloat(SUM(CASE WHEN eleito = 1 AND melhora = 0
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 1
  THEN 1 else 0 END))*10000)/100 AS eleitos_mantiveram,
  round(toFloat(SUM(CASE WHEN eleito = 0 AND melhora = 1
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 0
  THEN 1 ELSE 0 END))*10000)/100 AS nao_eleitos_1nivel,
  round(toFloat(SUM(CASE WHEN eleito = 1 AND melhora = 1
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 1
  THEN 1 else 0 END))*10000)/100 AS eleitos_1nivel,
  round(toFloat(SUM(CASE WHEN eleito = 0 AND melhora = 2
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 0
  THEN 1 else 0 END))*10000)/100 AS nao_eleitos_2_ou_mais,
  round(toFloat(SUM(CASE WHEN eleito = 1 AND melhora = 2
  THEN 1 ELSE 0 END))/toFloat(sum(CASE WHEN eleito = 1
  THEN 1 else 0 END))*10000)/100 AS eleitos_2_ou_mais
ORDER BY ano_troca ASC
```

Figura 5.10: Comparação entre melhoras nos níveis de escolaridade para candidatos eleitos *vs* não-eleitos, considerando-se a porcentagem sobre o total de candidatos.

```
CYPHER MATCH (el:Eleicoes)-->(a:Ano)-[:ano_candidatura]->(ca:Candidatura)-[:ref
```

ano_troca	nao_eleitos_mantiveram	eleitos_mantiveram	nao_eleitos_1nivel	eleitos_1nivel	nao_eleitos_2	eleitos_2
2004->2008	76.93	78.64	14.73	13.15	8.34	8.21
2008->2012	80.28	82.2	12.33	10.59	7.39	7.2

Fonte: Elaborado pelo autor.

A Consulta 20 é a mais longa, complexa e elaborada entre as executadas nestes experimentos, e seu resultado aparece na Figura 5.10. A primeira coluna da tabela apresentada se refere ao período analisado: primeiro, tem-se a evolução de escolaridade entre os anos 2004 e 2008 e, na segunda linha, entre os anos de 2008 e 2012. A segunda e a terceira colunas representam a porcentagem dos candidatos que mantiveram seu nível de escolaridade inalterado, primeiro para os não-eleitos e depois para os eleitos (em 2004, na primeira linha, e 2008, na segunda). Analogamente, a terceira e quarta colunas mostram a porcentagem de candidatos que melhorou 1 nível de escolaridade. A quinta e sexta coluna, fazem a mesma comparação, neste caso considerando a melhora de dois ou mais níveis de escolaridade.

Uma análise empírica pode sugerir que, com um cargo assalariado válido por quatro anos, um vereador pode atingir o aumento de sua renda e, consequentemente, fazer um maior investimento pessoal em educação. Partindo desta lógica, poderia-se assumir que candidatos eleitos melhoram, proporcionalmente, seus níveis de escolaridade ao longo dos anos. O resultado mostrado pela Consulta 20 mostra, entretanto, o exato oposto. Apesar do pequeno conjunto (em anos) dos dados analisado, pode-se inferir claramente que a maioria dos vereadores eleitos se mantem no mesmo nível de escolaridade ao longo do tempo, enquanto candidatos que não se elegeram, por manterem seu nível de vida, procuram investir em educação e melhoram seu grau de escolaridade.

Essa Consulta mostra, portanto, que, a partir do modelo criado neste trabalho, é possível fazer observações complexas e abrangentes a respeito não só dos dados eleitorais, como de aspectos que extrapolam o contexto das eleições para vereadores. Novamente, destaca-se que a adição de dados a esta base pode trazer um valor importante, permitindo diversos tipos de consultas, cruzando e agregando diversos tipos de dados e fazendo análises rebuscadas.

A Tabela 5.5 mostra, para as últimas consultas apresentadas, tempos que passam dos 3 segundos de execução. Esse resultado era esperado, uma vez que estas consultas são bastante complexas, envolve diversas agregações, além de cálculos matemáticos e con-

Tabela 5.5: Tempos de execução para outras consultas.

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 17	3.905	3.891 - 8.531
Consulta 18	3.991	3.984 - 7.359
Consulta 19	3.932	3.906 - 7.313
Consulta 20	5.668	5.656 - 10.047

Fonte: Elaborado pelo autor.

versões.

Por se tratarem de consultas mais estatísticas, que agregam e comparam informações gerais a respeito das eleições para vereador, não são consultas que um usuário executaria frequentemente e poderiam, em caso de necessidade, ser pré-processadas e armazenadas nos nodos correspondentes, conforme já sugerido anteriormente. O mais importante a se considerar aqui, entretanto, é o valor semântico dessas consultas. Elas revelam uma riqueza dos dados modelados, que se utilizados corretamente podem trazer informações úteis a respeito do cenário eleitoral, relevantes tanto para os partidos políticos como a população em geral. É por este valor agregado que, apesar dos altos tempos de execução, estas consultas ajudam a expressar o valor do modelo de dados criado neste trabalho.

5.3.6 Comparações Entre Consultas Cypher

Algumas consultas na linguagem Cypher podem ser escritas de duas ou mais formas diferentes, mas trazendo o mesmo resultado. O objetivo desta subseção é comparar algumas destas alternativas, justificando escolhas feitas para as consultas apresentadas anteriormente nesta seção.

Consulta 21. Total de candidaturas da cidade de São Paulo em 2008 usando relacionamento fortemente tipado.

```
MATCH (e:Eleicoes)-[:ano2008]->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(est:Estado)
WHERE est.sigla = 'sp' AND c.nome = 'sao paulo'
RETURN  count(ca) AS candidaturas_sao_paulo,
  c.nome AS cidade
```

Consulta 22. Total de candidaturas da cidade de São Paulo em 2008 usando pesquisa por atributo.

```
MATCH (el:Eleicoes)->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura)-[:pelo_municipio]->(c:Cidade)
  -[:pertence]->(est:Estado)
```

```
WHERE est.sigla = 'sp' AND c.nome = 'sao paulo'
  AND a.ano = 2008
RETURN count(ca) AS candidaturas_por_cidade,
  c.nome AS cidade"
```

Consulta 23. Total de candidaturas da cidade de São Paulo em 2008 usando pesquisa por atributo (descartando o nodo de referência Eleições).

```
MATCH (a:Ano)-[:ano_candidatura]->(ca:Candidatura)
  -[:pelo_municipio]->(c:Cidade)-[:pertence]->(est:Estado)
WHERE est.sigla = 'sp' AND c.nome = 'sao paulo'
  ANDa.ano = 2008
RETURN  count(ca) AS candidaturas_por_cidade,
  c.nome AS cidade
```

Tabela 5.6: Comparação entre consultas por ano instante no tempo.

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 21	100	94- 281
Consulta 22	86	78 - 234
Consulta 23	2.313	2.313 - 39.609

Fonte: Elaborado pelo autor.

A Tabela 5.6 mostra os tempos de execução para uma mesma consulta usando três alternativas diferentes. Pouca diferença foi observada entre a Consulta 21 e a Consulta 22, mas o resultado foi interessante, uma vez que em testes prévios não sistematizados, as consultas usando relacionamento fortemente tipados sempre apresentaram tempos de execução um pouco menor. Diversos fatores podem ter contribuído para essa interpretação, como o efeito de *cache* nas execuções das consultas. O fato é que, conforme já discutido anteriormente, o uso de relacionamentos fortemente tipado pode ser uma vantagem para um índice temporal mais hierárquico, como o do trabalho de Cattuto et al. (2013), que têm informação de ano, mês e dia. Entretanto, com uma diferença tão pequena na comparação com a consulta usando pesquisa por atributo, esta questão permanece em aberto, e sugere-se novamente a experimentação com índices mais completos (como o já proposto anteriormente, que combina índice temporal com índice geográfico).

A Consulta 23, entretanto, revela que a remoção do nodo *Eleicoes* trouxe uma degradação importante para o desempenho, principalmente se considerarmos o pior caso. Não foi realizado nenhum levantamento aprofundado sobre as causas desse impacto no tempo, que aparentemente não são intuitivas. Para evitar esse problema, portanto, utilizou-se o nodo *Eleicoes* em todas as consultas temporais, preferindo-se o filtro por relacionamento fortemente tipado em consultas por ano específico e filtrando pelo atributo em consultas

por faixas de tempo.

Consulta 24. Candidatos eleitos por partido no RS em 2004, filtrando pelos atributos via cláusula *WHERE*.

```
MATCH (el:Eleicoes)-[:ano2004]->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura )-[:pelo_municipio]
  ->(c:Cidade)-[:pertence]-(e:Estado),
  ca-[:pelo_partido]->(p:Partido)
WHERE ca.eleito = 1 ANDe.sigla = 'rs'
RETURN  count(ca) AS vereadores_por_partido,p.sigla
ORDER BY vereadores_por_partido DESC
```

Consulta 25. Candidatos eleitos por partido no RS em 2004, filtrando pelos atributos diretamente na seleção do nodo.

```
MATCH (el:Eleicoes)-[:ano2004]->(a:Ano)-[:ano_candidatura]
  ->(ca:Candidatura {eleito:  1})-[:pelo_municipio]
  ->(c:Cidade)-[:pertence]-(e:Estado {sigla:  'rs'}),
  ca-[:pelo_partido]->(p:Partido)
RETURN count(ca) AS vereadores_por_partido,p.sigla
ORDER BY vereadores_por_partido DESC
```

A linguagem Cypher permite aplicar restrições como as da cláusula *WHERE* diretamente no nodo declarado no padrão *MATCH*. A Tabela 5.8 mostra que, em termos de desempenho, não á diferença importante entre as duas alternativas.

A restrição usando cláusula `WHERE` foi adotada durante este trabalho por permitir maior clareza das consultas para um leitor acostumado com *SQL*, linguagem padrão em SGBDs relacionais. Entretanto, o filtro direto pelo nodo segue formato semelhante ao padrão JSON[13], amplamente utilizado na web. Isso tem o potencial de facilitar, em uma aplicação, o envio de filtros para uma determinada consulta `Cypher`. Portanto, considerou-se importante mencionar esta possibilidade.

Tabela 5.7: Comparação entre filtros por cláusula `WHERE` e restrição direta na seleção do nodo.

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 24	458	453 - 1.266
Consulta 25	462	453 - 1.250

Fonte: Elaborado pelo autor.

O conhecimento da estrutura do grafo permite ignorar os tipos dos relacionamentos nas consultas. Por exemplo, sabendo que existe a relação `(ca:Candidatura) ->`

[13] http://json.org/

`(c:Cidade)`, pode-se fazer consultas que incluam este padrão sem selecionar o tipo. A próxima comparação analisa o impacto de não incluir os tipos no padrão da consulta.

Consulta 26. Candidatos eleitos por partido no RS em 2004, ignorando os tipos de relacionamentos.

```
MATCH (el:Eleicoes)->(a:Ano)->(ca:Candidatura)
  ->(c:Cidade)-(e:Estado), (ca)->(p:Partido)
WHERE ca.eleito = 1 and e.sigla = 'rs' and a.ano = 2004
RETURN count(ca) as vereadores_por_partido,p.sigla
ORDER BY vereadores_por_partido DESC
```

Tabela 5.8: Comparação entre padrão com tipo de relacionamento e o mesmo padrão ignorando esses tipos.

Consulta	Tempo médio (10% - 90%)	Menor tempo - Maior tempo
Consulta 24	458	453 - 1.266
Consulta 26	510	500 - 27.582

Fonte: Elaborado pelo autor.

A Tabela 5.8 apresenta claramente o problema de ignorar os tipos dos relacionamentos na definição do padrão em uma consulta. Apesar do pouco impacto observado para os tempos médios, em comparação a uma consulta com todos os tipos de relacionamento, o que chama a atenção é o valor máximo: em um cenário sem cache, de pior caso, a consulta sem os tipos leva quase 30 segundos para ser executada. É preciso ter cuidado, portanto, com o tipo dos relacionamentos na definição das consultas, uma vez que ignorar um deles pode degradar consideravelmente a performance.

5.4 Considerações Finais

Os experimentos apresentados nesta Seção procuraram englobar os principais tipos de consultas temporais possíveis, de maneira a justificar que o modelo proposto atende aos requisitos temporais desejáveis, conforme os conceitos apresentados no Capítulo 2.

Consultas não temporais podem ser executadas normalmente, apenas ignorando-se o índice temporal. Acessar todas as informações relacionadas a um candidato, bem como pesquisar um candidato por nome, são operações triviais e eficientemente executadas.

É possível fazer consultas por um instante no tempo, ou seja, um ponto específico no índice temporal modelado. Durante a apresentação de consultas deste tipo, observou-se a degradação de desempenho para agregações que consideram um estado inteiro. Sugeriu-se como solução uma proposta para alterar o modelo, de maneira a incluir o índice geográfico vinculado ao índice temporal. Esta solução tem o potencial de trazer o benefício

de consultas por árvore de caminhamento, e é fortemente sugerida como trabalho futuro.

Também é trivial realizar consultas por períodos de tempo, e foram discutidas diversas maneiras de fazê-lo com tempos de execução aceitáveis.

O mais interessante, entretanto, são as consultas sobre evolução nos dados com o tempo. Estas consultas, juntamente com as apresentadas na subseção 5.3.5, revelam objetivamente o poder de expressão da modelagem temporal. É possível acessar o histórico de diversas características da base de dados, como o grau de escolaridade e profissão de um candidato a cada eleição, fazendo análises comparativas. Essas análises podem ser estendidas a casos mais abrangentes, de maneira a obter estatísticas complexas da evolução da informação, cruzando dados e fazendo agregações diversas, atingindo resultados semanticamente relevantes.

Esta seção ainda discutiu diversas maneiras de implementar consultas semelhantes, com seus prós e contras, mostrando a flexibilidade e versatilidade, tanto do SGBD Neo4j quanto da sua linguagem proprietária para consultas Cypher.

6 CONCLUSÃO

O presente trabalho modelou os dados referentes às três últimas eleições para vereador no Brasil, agregando mais de de um milhão de candidaturas em um banco de dados temporal. Desta maneira, foi possível analisar informações específicas sobre instantes do passado, períodos específicos no tempo e a evolução de informações ao longo do tempo, que permitiu levantar estatísticas importantes sobre o perfil dos candidatos.

Informações referentes a um candidato específico, ou mesmo totalizadores agregados para um determinado município selecionado, foram consultadas em tempos médios de execução aceitáveis. Analisa-se que, para um usuário comum, este tipo de consulta atende a todas as necessidades.

Entretanto, ao se considerar agregações mais complexas por volumes de dados consideravelmente maiores, o modelo atual apresenta algumas deficiências. Algumas consultas aparentemente simples tiveram tempos de execução acima de três segundos, o que seria inaceitável em uma aplicação web, por exemplo. Essa lentidão pode prejudicar análises mais complexas, como consultas considerando determinadas regiões, que incluem vários estados. Outras alternativas, algumas delas discutidas na apresentação dos resultados, devem ser consideradas em trabalho futuros.

Entretanto, o valor e importância do resultado dessas consultas permanece. Foi possível descobrir partidos que mudaram de nome ou foram extintos, a avaliação da participação feminina no cenário político do Rio Grande do Sul e relacionar o fator "candidato eleito"com níveis de escolaridade.

As análises das consultas sugerem uma lista considerável de possíveis trabalhos futuros. Como há informações disponível várias eleições passadas, seria possível aplicar o mesmo modelo a um número muito maior de eleições para vereador. Outros cargos também poderiam ser modelados, permitindo um mapeamento completo do perfil dos candidatos a cargos políticos no Brasil, bem como sua avaliação histórica.

O modelo deste trabalho se limitou as candidaturas, mas também existem dados disponíveis sobre declarações de bens de candidatos, número de votos votos e o perfil do eleitorado. Combinações dessas informações abrem um grande leque de possibilidade, com agregações e associações diversas.

Esses dados poderiam, ainda, ser exibidos de forma amigável ao usuário, utilizando-se por exemplo de refatoração, no lado do cliente, do resultado de consultas a motores de busca populares. Uma aplicação poderia, a partir de uma pesquisa pelo nome de um vereador no Google, consultar o banco de dados desenvolvidos neste trabalho e exibir todas as informações desse candidato, como seus dados ao longo dos anos ou candidatos do mesmo partido.

Consultas semânticas sobre bancos de dados de grafos temporais e a comparação de um modelo semelhante usando SGBD relacional também são sugeridas como trabalhos futuros.

REFERÊNCIAS

ANGLES, R.; GUTIERREZ, C. Survey of graph database models. **ACM Computing Surveys (CSUR)**, [S.l.], v.40, n.1, p.1, 2008.

CATTUTO, C. et al. Time-varying social networks in a graph database: a neo4j use case. In: FIRST INTERNATIONAL WORKSHOP ON GRAPH DATA MANAGEMENT EXPERIENCES AND SYSTEMS. **Anais...** [S.l.: s.n.], 2013. p.11.

EDELWEISS, N. Bancos de dados temporais: teoria e prática. In: XVII JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, DO XVIII CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. **Anais...** [S.l.: s.n.], 1998. v.2, p.225–282.

GUTIERREZ, C.; HURTADO, C. A.; VAISMAN, A. Introducing Time into RDF. **IEEE Trans. on Knowl. and Data Eng.**, Piscataway, NJ, USA, v.19, n.2, p.207–218, Feb. 2007.

KHURANA, U.; DESHPANDE, A. Efficient Snapshot Retrieval over Historical Graph Data. In: IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE 2013), 2013., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2013. p.997–1008. (ICDE '13).

LEAVITT, N. Will NoSQL databases live up to their promise? **Computer**, [S.l.], v.43, n.2, p.12–14, 2010.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to information retrieval**. [S.l.]: Cambridge university press Cambridge, 2008. v.1.

MCCOLL, R. et al. A brief study of open source graph databases. **arXiv preprint arXiv:1309.2675**, [S.l.], 2013.

NEO4J - The World's Leading Graph Database. Disponível em: http://www.neo4j.org. Data de Acesso: 10/07/2014.

NEO4J (CSV) Batch Importer. Disponível em: https://github.com/jexp/batch-import. Data de Acesso: 10/07/2014.

NEO4J Cypher Refcard 2.0. Disponível em: http://docs.neo4j.org/refcard/2.0. Data de Acesso: 10/07/2014.

NEO4J Rest Client. Disponível em: https://neo4j-rest-client.readthedocs.org. Data de Acesso: 10/07/2014.

PENTEADO, R. R. et al. Um Estudo sobre Bancos de Dados em Grafos Nativos. **X ERBD**, [S.l.], 2014.

RIZZOLO, F.; VAISMAN, A. A. Temporal XML: modeling, indexing, and query processing. **The VLDB Journal**, Secaucus, NJ, USA, v.17, n.5, p.1179–1212, Aug. 2008.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Databases**. [S.l.]: O'Reilly Media, Incorporated, 2013.