3
Modelo de Bloqueio Multigranular para RDF

Este capítulo, o principal deste documento, delineia o modelo de bloqueio proposto para transações sobre dados RDF. Trata-se de uma variante do protocolo multigranular exposto na seção Bloqueio Pessimista (2.1.2.1), que procura extrair da estrutura de triplas do RDF um protocolo de bloqueio pessimista que, ao mesmo tempo, aumente o nível de multiprogramação e proveja grânulos que estejam mais próximos dos conceitos de recurso (sujeito), propriedade (predicado) e valor (objeto) presentes no domínio RDF.

Ao definir um protococlo de bloqueio, para certo modelo de dados, devemos analisar a estrutura deste modelo, bem como as operações primitivas de manipulação dos dados (instâncias) descritos de acordo com este modelo. No caso do modelo RDF, a estrutura básica é um grafo composto de triplas do tipo "recurso-propriedade-valor" que descrevem as instâncias (recursos) e relacionamentos entre elas. Esta estrutura deve ser explorada para a definição de uma "hierarquia" de grânulos a serem bloqueados. Lembre-se que "hierarquia" significa a noção de um grânulo estar logicamente contido em outro, o que é importante para propagação de bloqueios. Como será abordado adiante neste capítulo, no caso do RDF, a "hierarquia" encontrada, na verdade, é um grafo acíclico, com uma única fonte (raiz). Quanto às operações, sabemos que em RDF não existem atualizações in-loco, ou seja, uma tripla não pode ser atualizada. Em RDF, *inserir* significa inserir triplas, *remover* significa remover triplas e *atualizar*, significa remover uma tripla e, em seguida, inserir uma nova tripla. Portanto, as operações primitivas de escrita em RDF são, logicamente e conceitualmente, *inserção* e *remoção*. Comparando com modelo relacional, por exemplo, em RDF, conceitualmente, não faz sentido a operação *atualização*, apenas *inserção*, *remoção* e, obviamente, *consulta*. Esta constatação óbvia foi levada em consideração e permitiu que novos tipos de bloqueio fossem criados, aumentando, em certos casos, o nível de multiprogramação quando comparado com os tipos de bloqueio de leitura e escrita convencionais.

Para definir um protocolo de bloqueio, quatro questões básicas precisam ser respondidas: *o que* bloquear e *como*, *quando* adquirir o bloqueio, quando

liberar o bloqueio, e como agir quando um bloqueio não puder ser adquirido. No modelo proposto, estas questões são respondidas como a seguir.

O *que* bloquear significa definir o grânulo. *Como* bloquear significa definir o tipo de bloqueio e o protocolo (regras) para bloquear os grânulos. Estas são as principais contribuições deste trabalho.

Quando obter o bloqueio, de acordo com a noção de prevenção da abordagem pessimista, significa bloquear o dado antes de usá-lo. Para garantir um escalonamento serializável, o protocolo de bloqueio em duas fases (2PL) tem que ser usado. Ainda, para evitar abortos em cascata, o protocolo *strict 2PL* deve ser empregado, o que significa liberar os bloqueios apenas ao final da transação.

Quando uma transação não consegue obter um bloqueio, uma opção bastante razoável é simplesmente abortar, esperar certo intervalo de tempo e tentar novamente. Para aliviar as consequências do aborto, a transação deve adquirir os bloqueios o mais cedo possível. Idealmente os bloqueios devem ser obtidos antes de o usuário começar seu trabalho (entrada de dados). Vale destacar que, uma vez que não há espera por bloqueios, este esquema é imune a *deadlocks* (negação da condição de *posse e espera*, necessária para ocorrência de *deadlocks*).

As seções seguintes descrevem os grânulos, os novos tipos de bloqueio e o protocolo de bloqueio proposto como um todo.

3.1. Grânulos

Para encontrar os grânulos em um nível de abstração apropriado para transações RDF, foram observadas quais porções de dados do modelo RDF uma transação usa, e como ela se refere a estes dados. Basicamente, uma transação manipula objetos (entidades) ou parte (propriedades) de objetos. Em RDF, a correspondência natural seria bloquear propriedades de recursos. Para os casos onde a transação acessa várias propriedades de um recurso, é mais simples bloquear o recurso inteiro, ao invés de bloquear cada propriedade individual. Para os casos onde a mesma propriedade de vários recursos é acessada, é mais simples bloquear a propriedade e, finalmente, para os casos onde várias propriedades de vários recursos são acessadas, o grafo inteiro pode ser bloqueado. Desta forma, transações longas não perdem tempo

estabelecendo um número exacerbado de bloqueios, e transações curtas não bloqueiam grandes porções do espaço de dados desnecessariamente.

A rigor, para encontrar os grânulos propostos, além das observações do parágrafo anterior, um processo mais sistemático de análise da tríade "*subjeito-predicado-objeto*" foi empregado. O processo consistiu em enumerar todas as possíveis configurações de triplas que poderiam ser bloqueadas, desde o grafo inteiro até uma única tripla, analisando cada uma delas de acordo com a abordagem pessimista de bloqueio. O passo a passo deste processo será apresentado a seguir.

Sejam dois recursos específicos *X* e *Y*, uma propriedade específica *P* e o termo *ALL*, significando todos os elementos. Há oito possíveis configurações de bloqueio, indo desde o grafo inteiro (granularidade máxima) até um *statement* único (granularidade mínima), apresentadas na Tabela 7.

Tabela 7 - Possíveis configurações "*Sujeito-Predicado-Objeto*"

Sujeito	Predicado	Objeto	Significado
ALL	**ALL**	**ALL**	Bloqueio no grafo todo.
ALL	**ALL**	**X**	Bloqueio em todas as propriedades do recurso X no papel de objeto.
ALL	**P**	**ALL**	Bloqueio na propriedade P para todos os recursos no papel de subjeito ou objeto.
ALL	**P**	**X**	Bloqueio na propriedade P do recurso X no papel de objeto.
X	**ALL**	**ALL**	Bloqueio em todas as propriedades do recurso X no papel de subjeito.
X	**ALL**	**Y**	Bloqueio em todos os *statements* que possuam o recurso X no papel de subjeito e o recurso Y no papel de objeto.
X	**P**	**ALL**	Bloqueio na propriedade P do recurso X no papel de subjeito.
X	**P**	**Y**	Bloqueio no *statement* específico X P Y. Ou seja, o *statement* que possua o recurso X no papel de subjeito, a propriedade P e o recurso Y no papel de objeto.

A seguir, na Tabela 8, a Tabela 7 é reescrita, porém substituindo-se os termos "*Sujeito-Predicado-Objeto*" por "*Recurso-Propriedade-Valor*" e

descrevendo-se a coluna *Significado* de acordo com os novos termos. Vale ressaltar que, apesar de mais intuitiva, a substituição da coluna "*Sujeito*" por "*Recurso*" e da coluna "*Objeto*" por "*Valor*" é uma escolha arbitrária. A mesma análise poderia ser feita substituindo a coluna "*Objeto*" por "*Recurso*" e a coluna "*Sujeito*" por "*Valor*" e chegaríamos as mesmas conclusões.

Tabela 8 - Possíveis configurações "*Recurso-Propriedade-Valor*"

Recurso	Propriedade	Valor	Significado
ALL	**ALL**	**ALL**	Todos os valores de todas as propriedades de todos os recursos.
ALL	**ALL**	**X**	Valor X de todas as propriedades de todos os recursos.
ALL	**P**	**ALL**	Todos os valores da propriedade P de todos os recursos.
ALL	**P**	**X**	Valor X da propriedade P de todos os recursos.
X	**ALL**	**ALL**	Todos os valores de todas as propriedades do recurso X.
X	**ALL**	**Y**	Valor Y de todas as propriedades do recurso X.
X	**P**	**ALL**	Todos os valores da propriedade P do recurso X.
X	**P**	**Y**	O valor Y da propriedade P do recurso X.

A Tabela 9 apresenta o mesmo conteúdo da Tabela 8, porém dando nomes aos grânulos candidatos na coluna *Significado*. Como será explicado mais adiante, com base no princípio básico da *abordagem* pessimista de *prevenção* de conflitos, os grânulos sublinhados são grânulos de fato. Já todos os grânulos que mencionam a palavra *valor* não são grânulos, são apenas valor do grânulo anterior na tabela (p. ex: Valor *X* na propriedade *P*, é um valor específico para o grânulo Propriedade P).

Tabela 9 - Grânulos Candidatos

Recurso	Propriedade	Valor	Significado
ALL	**ALL**	**ALL**	Grafo
ALL	**ALL**	**X**	Valor X no Grafo.
ALL	**P**	**ALL**	Propriedade P
ALL	**P**	**X**	Valor X na propriedade P.
X	**ALL**	**ALL**	Recurso X
X	**ALL**	**Y**	Valor Y no recurso X.
X	**P**	**ALL**	Propriedade P do Recurso x
X	**P**	**Y**	O valor Y na propriedade P do recurso X.

O princípio básico do bloqueio pessimista é prevenção de conflitos. Uma vez que prevenção significa a garantia que o usuário não perderá seu trabalho, o bloqueio tem que ser adquirido *antes* do trabalho do usuário começar. Por conseguinte, na tríade "*Recurso-Propriedade-Valor*" não podemos bloquear valores (quer sejam literais, quer sejam outros recursos), pois valores somente são conhecidos *depois* do trabalho do usuário, o que vai de encontro ao princípio pessimista. Sem considerar que bloquear valores geraria bloqueios em nível de triplas, o que seria um *overhead* bem alto.

Sendo assim, para obter os verdadeiros grânulos, temos que eliminar a coluna *Valor* da Tabela 9, resultando na Tabela 10 e no *Lock Type Graph da* Figura 27 (já com os termos em inglês, utilizados na implementação). Os grânulos "*Recurso"*, "*Propriedade"* e "*Propriedade de Recurso"* nada mais são do que as URIs envolvidas, que, respectivamente, são: (*URI do Recurso*), (*URI da Propriedade*) e (*URI da Propriedade, URI do Recurso*). Já o grânulo "*Grafo"*, não possui URI associada, pois significa bloquear tudo.

Tabela 10 - Grânulos para transações Web RDF

Recurso	Propriedade	Significado
ALL	**ALL**	Grafo
ALL	**P**	Propriedade P
X	**ALL**	Recurso X
X	**P**	Propriedade P do Recurso X

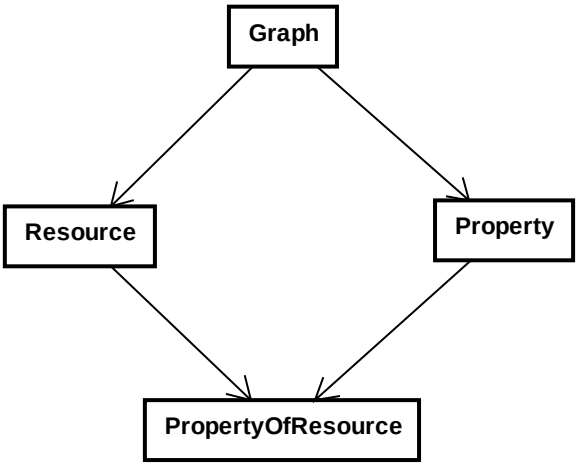

Figura 27 - *Rooted DAG[42] Lock Type Graph*

É importante ressaltar que ao bloquear um grânulo que envolva uma propriedade *P* ("*PropertyOfResource*" ou "*Property*"), tem que bloquear também o grânulo "*Property*" da propriedade inversa P_{inv}, caso exista, para garantir que o bloqueio não seja violado no sentido inverso. Ou seja, para estes tipos de grânulo, a transação, ao requisitar o bloqueio ao *Lock Manager*, tem que passar como argumento ambas as propriedades *P* e P_{inv}. Por exemplo, se uma transação T_1 bloquear a propriedade "*ex:leciona*" do recurso "*ex:Schwabe*", então tem que bloquear a propriedade "*ex:lecionadaPor*" com o mesmo tipo de bloqueio. Isto para evitar que o bloqueio seja violado no sentido contrário, como por exemplo, no caso de uma segunda transação T_2 que tente bloquear a propriedade "*ex:lecionadaPor*" do recurso "*ex:WebSemantica*" com o intuito de inserir o valor "*ex:Schwabe*", o que implicaria na inserção no sentido contrário "*ex:Schwabe*" "*ex:leciona*" "*ex:WebSemantica*".

3.2. Tipos de Bloqueio

Para os tipos de bloqueio (*lock modes*, em inglês), a ideia é explorar o aspecto multivalorado das propriedades RDF. Desta forma, os quatro grânulos propostos podem ser bloqueados em seis modos: *removal read lock (rR), insertion read lock (iR), removal/insertion read lock (riR), removal write lock (rW), insertion write lock (iW) e removal/insertion write lock (riW)*. Todos são sumarizados na Tabela 11. Nesta tabela, cada par ordenado (linha, coluna)

corresponde a um tipo de bloqueio cujo significado é explicado na célula (linha, coluna) correspondente.

Tabela 11 - Tipos de bloqueio (*lock modes*) propostos

	Write Lock	***Read Leitura***
Removal	Deve ser usado quando o objetivo é apenas remover *statements*.	Deve ser usado quando o objetivo é apenas impedir remoções de *statements*.
Insertion	Deve ser usado quando o objetivo é apenas inserir *statements*.	Deve ser usado quando o objetivo é apenas impedir inserções de *statements*.
Remoção/ Inserção	Deve ser usado quando o objetivo é remover e inserir *statements*.	Deve ser usado quando o objetivo é impedir remoções e inserções de *statements*.

A Tabela 12 mostra a matriz de compatibilidade entre os tipos de bloqueio propostos. Um "s" significa "Sim", ou seja, para certo grânulo, os dois tipos de bloqueio podem ser adquiridos, ao mesmo tempo, por transações distintas. Um "n" significa "Não", ou seja, um tipo de bloqueio impede que o outro seja adquirido por uma segunda transação, sobre o mesmo grânulo (conflito). Diferentemente dos tipos de bloqueio tradicionais, onde bloqueio de escrita é completamente exclusivo, nesta proposta, existem casos onde é possível estabelecer bloqueios de escrita e leitura, por transações diferentes, no mesmo item de dado, ao mesmo tempo. Isto porque um bloqueio pode ser para inserção e o outro para remoção. Por exemplo, os tipos de bloqueio *rR* e *iW* são compatíveis, pois o primeiro impede apenas remoções de triplas e o segundo é usado apenas para inserções de triplas. Raciocínio similar explica a compatibilidade entre os tipos de bloqueio *iR* e *rW*. No entanto, bloqueios de escrita, em todos os modos, são mutuamente exclusivos entre si, uma vez que modificações, por definição, são sempre feitas com base no estado original.

[42] *Rooted DAG* - Grafo Acíclico Direcionado com uma única fonte chamada raiz. Fonte é um nó sem arestas de entrada.

Tabela 12 - Compatibilidade entre os tipos de bloqueio

	rR	**iR**	**riR**	**rW**	**iW**	**riW**
rR	s	s	s	n	s	n
iR	s	s	s	s	n	n
riR	s	s	s	n	n	n
rW	n	s	n	n	n	n
iW	s	n	n	n	n	n
riW	n	n	n	n	n	n

Existem quatro de problemas comuns de concorrência: *lost updates, dirty reads, non-repeatable reads e phantom reads*. No contexto de transações RDF, e utilizando os novos tipos de bloqueio propostos, cada um destes problemas pode ser prevenido da seguinte forma:

- ***Lost Updates***

  Em RDF, seriam duas ou mais transações selecionarem o mesmo recurso e então atualizar o recurso, inserindo/removendo *statements*, com base no estado original (*statements* originais), sem tomar conhecimento das atualizações concorrentes. Isto resultaria num conjunto final de *statements* inconsistentes, uma vez que não foram observados os *statements* inseridos/removidos por transações concorrentes. Para impedir este problema, basta a transação adquirir um dos bloqueios de escrita. Obviamente, todos os bloqueios de escrita são mutuamente exclusivos entre si.

- ***Dirty Reads***

  Em RDF, seria uma segunda transação "enxergar" *statements* não confirmados (*uncommitted*) de outra transação concorrente. Uma vez que não existe atualização in-loco de um *statement* RDF, na verdade, temos que evitar que os *statements* lidos sejam removidos. Para prevenir o problema, basta a transação adquirir um bloqueio de leitura para remoção (*rR*) ou mais forte, durante a operação de leitura.

- ***Non-repeatable Reads***

  Em RDF, seria uma segunda transação acessar os mesmos *statements* mais de uma vez e lê dados diferentes a cada vez. Uma

vez que não existe atualização in-loco de um *statement* RDF, na verdade, temos que evitar que os *statements* lidos sejam removidos. Para prevenir o problema, basta a transação adquirir um bloqueio de leitura para remoção (*rR*) ou mais forte, até o final da transação.

- ***Phantom Reads***

  Em RDF, seria um *statement* pertencente a uma coleção lida por uma transação, ser removido ou inserido por uma segunda transação. Para evitar o problema, basta a transação adquirir um bloqueio de leitura para remoção/inserção (*riR*) ou mais forte, até o final da transação. Ou seja, impedir remoção e inserção de *statements* RDF.

3.3. Protocolo de Bloqueio Multigranular

Conforme descrito na seção 2.1.2.1 (Bloqueio Pessimista), um protocolo de bloqueio multigranular requer um *Lock Manager* que previna duas transações de adquirirem bloqueios conflitantes em dois grânulos que se sobreponham. Este é o critério de corretude. Por exemplo, um recurso não deve ser bloqueado para escrita por uma transação T1, se uma propriedade deste recurso estiver bloqueada para escrita por outra transação T2. Uma solução insatisfatória seria requerer que a transação T1 verificasse cada propriedade do recurso para ver se pode bloquear o recurso. Isto iria de encontro ao propósito de bloquear um grânulo maior que é reduzir o *overhead* de boqueio.

Uma solução melhor é explorar o relacionamento hierárquico existente entre os diferentes grânulos. Estes relacionamentos são representados pelo *lock type graph da* Figura 27, onde uma aresta conecta um tipo de dados de maior granularidade a um de menor granularidade. Assim, um bloqueio em um grânulo *x*, *explicitamente* bloqueia *x* e *implicitamente* bloqueia todos os descendentes de *x*, ou seja, grânulos menores logicamente contidos em x. Por exemplo, um bloqueio em um recurso implicitamente bloqueia todas propriedades do recurso.

No sentido inverso, também é necessário propagar o efeito de bloquear um grânulo menor para os grânulos maiores que logicamente o contenham. Para fazer isto, para cada tipo de bloqueio apresentado anteriormente, existe um correspondente bloqueio planejado (*planned lock*), também conhecido como

bloqueio de intenção (*intention lock*) na literatura (vide seção 2.1.2.1). Por exemplo, para o *Bloqueio de Leitura para Remoção* (*rR*) existe o *Bloqueio Planejado de Leitura para Remoção* (*prR*). *Antes* de bloquear *x*, o *Lock Manager* tem que garantir que não existam bloqueios nos ancestrais de *x* que *implicitamente* estabeleçam um bloqueio conflitante em *x*. Por exemplo, para estabelecer um *Bloqueio de Leitura para Remoção* no recurso *x rRL[x]*, antes é estabelecido um *Bloqueio Planejado de Leitura para Remoção* em todo o grafo *prRL[graph]*, que é o único ancestral neste caso. Estabelecendo apropriadamente bloqueios planejados nos ancestrais de *x*, o *Lock Manager* garante que não haverá bloqueio que implicitamente bloqueie *x* em modo conflitante. A Tabela 13 apresenta uma nova matriz de compatibilidade, estendida com os correspondentes bloqueios planejados (iniciados com *p*, na tabela). Perceba que, com respeito à compatibilidade com outro bloqueio (real), cada bloqueio planejado tem o mesmo comportamento do seu correspondente bloqueio (real). No entanto, quaisquer bloqueios planejados são sempre compatíveis entre si, afinal são apenas planejados. Por exemplo, o bloqueio planejado *prR* segue seu correspondente bloqueio real *rR*, sendo incompatível com *rW* e *riW*. No entanto, *prR* é compatível com *prW* e *priW*. Isto é perfeitamente coerente com o papel de *prR* que é apenas evitar que seja estabelecido um bloqueio real incompatível com *rR*.

Tabela 13 - Compatibilidade entre os tipos de bloqueio, incluindo bloqueios planejados

	rR	**iR**	**riR**	**rW**	**iW**	**riW**	**prR**	**piR**	**priR**	**prW**	**piW**	**priW**
rR	s	s	s	n	s	n	s	s	s	n	s	n
iR	s	s	s	s	n	n	s	s	s	s	n	n
riR	s	s	s	n	n	n	s	s	s	n	n	n
rW	n	s	n	n	n	n	n	s	n	n	n	n
iW	s	n	n	n	n	n	s	n	n	n	n	n
riW	n	n	n	n	n	n	n	n	n	n	n	n
prR	s	s	s	n	s	n	s	s	s	s	s	s
piR	s	s	s	s	n	n	s	s	s	s	s	s
riR	s	s	s	n	n	n	s	s	s	s	s	s
prW	n	s	n	n	n	n	s	s	s	s	s	s
piW	s	n	n	n	n	n	s	s	s	s	s	s
priW	n	n	n	n	n	n	s	s	s	s	s	s

O protocolo MGL da seção 2.1.2.1, que assume que a hierarquia de grânulos é uma árvore, pode ser expandido para estruturas mais gerais como um *lock instance graph* G estruturado de acordo com o *rooted* DAG da Figura

27. Dado que se trata de um *rooted* DAG, pode existir mais de um caminho da raiz até certo item de dado, o que significa que o item de dado pode ter mais de um pai que o bloqueie implicitamente. Para evitar que o item de dado seja implicitamente bloqueado por bloqueios conflitantes, oriundos de pais distintos, é necessário propagar os bloqueios de escrita para todos os pais. Quanto aos bloqueios de leitura, basta propagar para um dos pais, dado que bloqueio de leitura somente pode ter conflito com outro bloqueio de escrita (nunca com outro de leitura) e bloqueio de escrita sempre tem conflito com bloqueio de escrita. Desta forma, não haverá como se configurar dois bloqueios conflitantes implícitos no item de dado.

Utilizando a extensão descrita no parágrafo anterior e os novos tipos de bloqueio aqui propostos, o *Lock Manager* estabelece e libera bloqueios para cada transação T_i de acordo com o seguinte protocolo:

1. Se *x* não for a raiz de G, então para estabelecer *rRL$_i$[x]* ou *prRL$_i$[x]*, T_i tem que ter um bloqueio *prR* ou *priR* ou *prW* ou *piW* ou *priW* em algum pai de *x*.
2. Se *x* não for a raiz de G, então para estabelecer *iRL$_i$[x]* ou *piRL$_i$[x]*, T_i tem que ter um bloqueio *piR* ou *priR* ou *prW* ou *piW* ou *priW* em algum pai de *x*.
3. Se *x* não for a raiz de G, então para estabelecer *riRL$_i$[x]* ou *priRL$_i$[x]*, T_i tem que ter um bloqueio *priR* ou *prW* ou *piW* ou *priW* em algum pai de x.
4. Se *x* não for a raiz de G, então para estabelecer *rWL$_i$[x]* ou *prWL$_i$[x]*, T_i tem que ter um bloqueio *prW* ou *priW* em todos os pais de x.
5. Se *x* não for a raiz de G, então para estabelecer *iWL$_i$[x]* ou *piWL$_i$[x]*, T_i tem que ter um bloqueio *piW* ou *priW* em todos os pais de *x*.
6. Se não for a raiz de G, então para estabelecer *riWL$_i$[x]* ou *priWL$_i$[x]*, T_i tem que ter um bloqueio *priW* em todos os pais de x.
7. Para ler *x*, T_i tem que ter um bloqueio de leitura (*r*) ou um bloqueio de escrita (*w*) em algum ancestral de *x*. Para escrever *x*, T_i tem que ter, para todos os caminhos da raiz de G até *x*, um bloqueio de escrita (*w*) em algum ancestral de *x* ao longo de cada caminho. Um bloqueio em *x* é um bloqueio *explícito* de *x*; bloqueios sobre acenstrais de *x* são bloqueios *implícitos* de *x*.
8. Uma transação não pode liberar um bloqueio planejado em um item *x*, se a transação possuir correntemente um bloqueio em pelo menos um filho de *x*.

Similar ao protocolo MGL apresentado na seção 2.1.2.1, as regras de (1) a (6) implicam que para a transação T_i adquirir um bloqueio *L[x]*, T_i tem que primeiro adquirir bloqueios planejados (ou de intenção) apropriados nos ancestrais de *x*, ou seja, bloqueios devem ser obtidos na ordem raiz-para-folha. Regra (7) implica que bloqueando *x*, T_i bloqueia, implicitamente, todos os descendentes de *x* em G. Este bloqueio implícito evita que a transação tenha que adquirir bloqueios nos descendentes de *x*, o que é a principal razão de um esquema de bloqueio multigranular. Regra (8) implica que bloqueios devem ser liberados na ordem folha-para-raiz, ou seja, sentido inverso da ordem em que os bloqueios foram adquiridos.

3.4. Corretude do Protocolo

Inspirado pela prova do protocolo MGL da seção 2.1.2.1, descrita em [Bernstein et al., 1987], esta seção apresenta a prova de corretude do protocolo proposto.

O objetivo do protocolo anterior é garantir que transações nunca adquiram bloqueios conflitantes (explícitos ou implícitos) no mesmo item de dado (ou seja, nó do *Lock Type Graph*).

Teorema

Suponha que todas as transações obedeçam o referido protocolo com relação a um dado *Lock Type Graph*, G, que seja *rooted DAG* (como da Figura 27). Se uma transação possuir um bloqueio (explícito ou implícito) em um nó de G, então nenhuma outra transação possuirá um bloqueio conflitante (explícito ou implícito) naquele nó.

Prova

É suficiente provar o teorema para os nós folha. Isto porque se duas transações possuíssem bloqueios conflitantes (explícitos ou implícitos) em um nó interno (não folha) *x*, elas possuiriam bloqueios conflitantes (implícitos) em todos os descendentes e, em particular, em todas as folhas descendentes de *x*. Suponha então que as transações T_i e T_j possuam bloqueios conflitantes na folha *x*. Existem 49 (quarenta e nove) casos possíveis, devidamente negados a seguir:

1. **Bloqueio rR implícito, Bloqueio rW explícito**:
   Pela regra (7), T_i tem $rRL_i[y]$ em algum ancestral y de x. Pela regra (4) e por indução, T_j tem um bloqueio prW ou priW em todos os ancestrais de x. Em particular, T_j tem $prWL_j[y]$ ou $priWL_j[y]$, o que conflita com $rRL_i[y]$.

2. **Bloqueio rR implícito, Bloqueio rW implícito**:
   Pela regra (7), T_i tem $rRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $rWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (4) e por indução, T_j tem $prWL_j[y]$ ou $priWL_j[y]$, o que conflita com $rRL_i[y]$. E o caso (c) é impossível porque, pela regra (1) e por indução, T_i tem $prRL_i[y']$ ou $priRL_i[y']$, o que conflita com $rWL_j[y']$.

3. **Bloqueio rR implícito, Bloqueio riW explícito**:
   Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $rRL_i[y]$ em algum ancestral y de x. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x. Em particular, T_j tem $priWL_j[y]$, o que conflita com $rRL_i[y]$.

4. **Bloqueio rR implícito, Bloqueio riW implícito**:
   Segue o mesmo raciocínio do caso 2. Pela regra (7), T_i tem $rRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $rRL_i[y]$. E o caso (c) é impossível porque, pela regra (1) e por indução, T_i tem $prRL_i[y']$ ou $priRL_i[y']$, o que conflita com $riWL_j[y']$.

5. **Bloqueio rR explícito, Bloqueio rW explícito**:
   Claramente impossível.

6. **Bloqueio rR explícito, Bloqueio rW implícito**:
   Pela regra (1) e por indução, T_i tem um bloqueio prR ou priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $rWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $prRL_i[y]$ e $priRL_i[y]$.

7. **Bloqueio rR explícito, Bloqueio riW explícito**:
   Claramente impossível.

8. **Bloqueio rR explícito, Bloqueio riW implícito**:
   Segue o mesmo raciocínio do caso 6. Pela regra (1) e por indução, T_i tem um bloqueio prR ou priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $prRL_i[y]$ e $priRL_i[y]$.

9. **Bloqueio iR implícito, Bloqueio iW explícito**:
   Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $iRL_i[y]$ em algum ancestral y de x. Pela regra (5) e por indução, T_j tem um bloqueio piW ou priW em todos os ancestrais de x. Em particular, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $iRL_i[y]$.

10. **Bloqueio iR implícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 2. Pela regra (7), T_i tem $iRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $iWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (5) e por

indução, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $rRL_i[y]$. E o caso (c) é impossível porque, pela regra (2) e por indução, T_i tem $piRL_i[y']$ ou $priRL_i[y']$, o que conflita com $iWL_j[y']$.

11. **Bloqueio iR implícito, Bloqueio riW explícito**:
    Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $iRL_i[y]$ em algum ancestral y de x. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x. Em particular, T_j tem $priWL_j[y]$, o que conflita com $iRL_i[y]$.

12. **Bloqueio iR implícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 2. Pela regra (7), T_i tem $iRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $iRL_i[y]$. E o caso (c) é impossível porque, pela regra (2) e por indução, T_i tem $piRL_i[y']$ ou $priRL_i[y']$, o que conflita com $riWL_j[y']$.

13. **Bloqueio iR explícito, Bloqueio iW explícito**:
    Claramente impossível.

14. **Bloqueio iR explícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 6. Pela regra (2) e por indução, T_i tem um bloqueio piR ou priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $iWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $piRL_i[y]$ e $priRL_i[y]$.

15. **Bloqueio iR explícito, Bloqueio riW explícito**:
    Claramente impossível.

16. **Bloqueio iR explícito, Bloqueio riW implícito**:

    Segue o mesmo raciocínio do caso 6. Pela regra (2) e por indução, T_i tem um bloqueio piR ou priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $piRL_i[y]$ e $priRL_i[y]$.

17. **Bloqueio riR implícito, Bloqueio rW explícito**:

    Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $riRL_i[y]$ em algum ancestral y de x. Pela regra (4) e por indução, T_j tem um bloqueio prW ou priW em todos os ancestrais de x. Em particular, T_j tem $prWL_j[y]$ ou $priWL_j[y]$, o que conflita com $riRL_i[y]$.

18. **Bloqueio riR implícito, Bloqueio rW implícito**:

    Segue o mesmo raciocínio do caso 2. Pela regra (7), T_i tem $riRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $rWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (4) e por indução, T_j tem $prWL_j[y]$ ou $priWL_j[y]$, o que conflita com $riRL_i[y]$. E o caso (c) é impossível porque, pela regra (3) e por indução, T_i tem $priRL_i[y']$, o que conflita com $rWL_j[y']$.

19. **Bloqueio riR implícito, Bloqueio iW explícito**:

    Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $riRL_i[y]$ em algum ancestral y de x. Pela regra (5) e por indução, T_j tem um bloqueio piW ou priW em todos os ancestrais de x. Em particular, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $riRL_i[y]$.

20. **Bloqueio riR implícito, Bloqueio iW implícito**:

    Segue o mesmo raciocínio do caso 2. Pela regra (7) , T_i tem $riRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho.

Em particular, T_j tem $iWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (5) e por indução, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $riRL_i[y]$. E o caso (c) é impossível porque, pela regra (3) e por indução, T_i tem $priRL_i[y']$, o que conflita com $iWL_j[y']$.

21. **Bloqueio riR implícito, Bloqueio riW explícito**:
    Segue o mesmo raciocínio do caso 1. Pela regra (7), T_i tem $riRL_i[y]$ em algum ancestral y de x. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x. Em particular, T_j tem $priWL_j[y]$, o que conflita com $riRL_i[y]$.

22. **Bloqueio riR implícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 2. Pela regra (7), T_i tem $riRL_i[y]$ em algum ancestral y de x, e T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y']$ em algum ancestral y' ao longo do caminho onde se encontra o ancestral y. Neste caso há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de leitura e escrita, respectivamente. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $riRL_i[y]$. E o caso (c) é impossível porque, pela regra (3) e por indução, T_i tem $priRL_i[y']$, o que conflita com $riWL_j[y']$.

23. **Bloqueio riR explícito, Bloqueio rW explícito**:
    Claramente impossível.

24. **Bloqueio riR explícito, Bloqueio rW implícito**:
    Segue o mesmo raciocínio do caso 6. Pela regra (3) e por indução, T_i tem um bloqueio priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada

caminho. Em particular, T_j tem $rWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $priRL_i[y]$.

25. **Bloqueio riR explícito, Bloqueio iW explícito**:
    Claramente impossível.

26. **Bloqueio riR explícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 6. Pela regra (3) e por indução, T_i tem um bloqueio priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $iWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $priRL_i[y]$.

27. **Bloqueio riR explícito, Bloqueio riW explícito**:
    Claramente impossível.

28. **Bloqueio riR explícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 6. Pela regra (3) e por indução, T_i tem um bloqueio priR em todos ancestrais de x, ao longo de algum caminho c da raiz de G até x. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Em particular, T_j tem $riWL_j[y]$ em algum ancestral y de x ao longo do caminho c, o que conflita com $priRL_i[y]$.

29. **Bloqueio rW implícito, Bloqueio rW explícito**:
    Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (4) e por indução, T_j tem um bloqueio prW ou priW em todos os ancestrais de x, o que conflita com o bloqueio rW de algum ancestral de x ao longo de cada caminho de G até x.

30. **Bloqueio rW implícito, Bloqueio rW implícito**:
    Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Em cada caminho da raiz G até x, T_i tem $rWL_i[y]$ em algum ancestral y de x e T_j tem $rWL_j[y']$ em algum ancestral y' de x. Assim, há três subcasos: (a) y =

y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (4) e por indução, T_j tem $prWL_j[y]$ ou $priWL_j[y]$, o que conflita com $rWL_i[y]$. E o caso (c) é impossível porque, pela regra (4) e por indução, T_i tem $prWL_i[y']$ ou $priWL_i[y']$, o que conflita com $rWL_j[y']$.

31. **Bloqueio rW implícito, Bloqueio iW explícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (5) e por indução, T_j tem um bloqueio piW ou priW em todos os ancestrais de x, o que conflita com o bloqueio rW de algum ancestral de x ao longo de cada caminho de G até x.
32. **Bloqueio rW implícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 30. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Em cada caminho da raiz G até x, T_i tem $rWL_i[y]$ em algum ancestral y de x e T_j tem $iWL_j[y']$ em algum ancestral y' de x. Assim, há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (5) e por indução, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $rWL_i[y]$. E o caso (c) é impossível porque, pela regra (4) e por indução, T_i tem $prWL_i[y']$ ou $priWL_i[y']$, o que conflita com $iWL_j[y']$.

33. **Bloqueio rW implícito, Bloqueio riW explícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x, o que conflita com o bloqueio rW de algum ancestral de x ao longo de cada caminho de G até x.

34. **Bloqueio rW implícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 30. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio rW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho.Em cada caminho da raiz G até x, T_i tem $rWL_i[y]$ em algum ancestral y de x e T_j tem $riWL_j[y']$ em algum ancestral y' de x. Assim, há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $rWL_i[y]$. E o caso (c) é impossível porque, pela regra (4) e por indução, T_i tem $prWL_i[y']$ ou $priWL_i[y']$, o que conflita com $riWL_j[y']$.

35. **Bloqueio rW explícito, Bloqueio rW explícito**:
    Claramente impossível.

36. **Bloqueio rW explícito, Bloqueio iW explícito**:
    Claramente impossível.

37. **Bloqueio rW explícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Pela regra (4) e por indução, T_i tem um bloqueio prW ou priW em todos os ancestrais de x, o que conflita com o bloqueio iW de algum ancestral de x ao longo de cada caminho de G até x.

38. **Bloqueio rW explícito, Bloqueio riW explícito**:
    Claramente impossível.

39. **Bloqueio rW explícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Pela regra (4) e por indução, T_i tem um bloqueio prW ou priW em todos os ancestrais de x, o que conflita com o

bloqueio riW de algum ancestral de x ao longo de cada caminho de G até x.

40. **Bloqueio iW implícito, Bloqueio iW explícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Pela regra (5) e por indução, T_j tem um bloqueio piW ou priW em todos os ancestrais de x, o que conflita com o bloqueio iW de algum ancestral de x ao longo de cada caminho de G até x.

41. **Bloqueio iW implícito, Bloqueio iW implícito**:
    Segue o mesmo raciocínio do caso 30. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho.Em cada caminho da raiz G até x, T_i tem $iWL_i[y]$ em algum ancestral y de x e T_j tem $iWL_j[y']$ em algum ancestral y' de x. Assim, há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (5) e por indução, T_j tem $piWL_j[y]$ ou $priWL_j[y]$, o que conflita com $iWL_i[y]$. E o caso (c) é impossível porque, pela regra (5) e por indução, T_i tem $piWL_i[y']$ ou $priWL_i[y']$, o que conflita com $iWL_j[y']$.

42. **Bloqueio iW implícito, Bloqueio riW explícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x, o que conflita com o bloqueio iW de algum ancestral de x ao longo de cada caminho de G até x.

43. **Bloqueio iW implícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 30. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio iW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao

longo de cada caminho. Em cada caminho da raiz G até x, T_i tem $iWL_i[y]$ em algum ancestral y de x e T_j tem $riWL_j[y']$ em algum ancestral y' de x. Assim, há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $iWL_i[y]$. E o caso (c) é impossível porque, pela regra (5) e por indução, T_i tem $piWL_i[y']$ ou $priWL_i[y']$, o que conflita com $riWL_j[y']$.

44. **Bloqueio iW explícito, Bloqueio iW explícito**:
    Claramente impossível.

45. **Bloqueio iW explícito, Bloqueio riW explícito**:
    Claramente impossível.

46. **Bloqueio iW explícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Pela regra (5) e por indução, T_i tem um bloqueio piW ou priW em todos os ancestrais de x, o que conflita com o bloqueio riW de algum ancestral de x ao longo de cada caminho de G até x.

47. **Bloqueio riW implícito, Bloqueio riW explícito**:
    Segue o mesmo raciocínio do caso 29. Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Pela regra (6) e por indução, T_j tem um bloqueio priW em todos os ancestrais de x, o que conflita com o bloqueio riW de algum ancestral de x ao longo de cada caminho de G até x.

48. **Bloqueio riW implícito, Bloqueio riW implícito**:
    Segue o mesmo raciocínio do caso 30:Pela regra (7), T_i tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho. Pela regra (7), T_j tem, para todos os caminhos da raiz de G até x, um bloqueio riW em algum ancestral de x ao longo de cada caminho.Em cada caminho da raiz G até x, T_i tem $riWL_i[y]$ em algum ancestral y de x e T_j tem $riWL_j[y']$ em algum ancestral y' de x.

Assim, há três subcasos: (a) y = y'; (b) y é um ancestral de y'; (c) y' é um ancestral de y. Caso (a) é impossível porque T_i e T_j têm bloqueios conflitantes de escrita e escrita. Caso (b) é impossível porque, pela regra (6) e por indução, T_j tem $priWL_j[y]$, o que conflita com $riWL_i[y]$. E o caso (c) é impossível porque, pela regra (6) e por indução, T_i tem $priWL_i[y']$, o que conflita com $riWL_j[y']$.

49. **Bloqueio riW explícito, Bloqueio riW explícito**:
Claramente impossível.

3.5. Conversão de Bloqueios

Seja um item de dado *x*. Se uma transação T_i já possui um bloqueio $mL_i[x]$ e requisita um bloqueio $nL_i[x]$, diretamente ou por propagação, então o *Lock Manager* deve converter (*upgrade*) $mL_i[x]$ em um tipo de bloqueio que seja pelo menos tão forte quanto ambos *m* e *n*. Um exemplo: se *m* = *rR* e *n* = *rW*, então o *Lock Manager* deve converter $rRL_i[x]$ em $rWL_i[x]$. Neste exemplo basta apenas o tipo de bloqueio *rW*, pois um bloqueio de escrita é sempre mutuamente exclusivo com outro bloqueio de escrita, garantido, obviamente, qualquer bloqueio de leitura. Outro exemplo, agora envolvendo propagação: se *m* = *rR* e *n* = *prW*, então o *Lock Manager* deve converter $rRL_i[x]$ em $rRprWL_i[x]$. Perceba que neste último exemplo surgiu um tipo de bloqueio composto, *rRprW*, resultante da combinação dos tipos de bloqueio primitivos *rR* e *prW*. Ou seja, não existe um tipo de bloqueio primitivo que seja forte o suficiente para atender ambos os bloqueios, portanto tem que ser criado um tipo de bloqueio resultante da combinação dos dois tipos primitivos de bloqueio. A Tabela 14, apresenta todas as possíveis conversões (entre bloqueios primitivos). Em azul, estão os bloqueios compostos resultantes.

Tabela 14 - Matriz de conversão entre bloqueios primitivos

		Requested Lock Type											
		rR	iR	riR	rW	iW	riW	prR	piR	priR	prW	piW	priW
Old Lock Type	rR	rR	riR	riR	rW	iW	riW	rR	rRpiR	rRpiR	rRprW	rRpiW	rRpriW
	iR	riR	iR	riR	rW	iW	riW	iRprR	iR	iRprR	iRprW	iRpiW	iRpriW
	riR	riR	riR	riR	rW	iW	riW	riR	riR	riR	riRprW	riRpiW	riRpriW
	rW	rW	rW	rW	rW	riW	riW	rW	rW	rW	rW	rWpiW	rWpiW
	iW	iW	iW	iW	riW	iW	riW	iW	iW	iW	iWprW	iW	iWprW
	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW
	prR	rR	iRprR	riR	rW	iW	riW	prR	priR	priR	prW	piW	priW
	piR	rRpiR	iR	riR	rW	iW	riW	priR	piR	priR	prW	piW	priW
	priR	rRpiR	iRprR	riR	rW	iW	riW	priR	priR	priR	prW	piW	priW
	prW	rRprW	iRprW	riRprW	rW	iWprW	riW	prW	prW	prW	prW	priW	priW
	piW	rRpiW	iRpiW	riRpiW	rWpiW	iW	riW	piW	piW	piW	priW	piW	priW
	priW	rRpriW	iRpriW	riRpriW	rWpiW	iWprW	riW	priW	priW	priW	priW	priW	priW

Para efeitos de implementação, ainda precisa ser definida uma versão final das matrizes de compatibilidade e de conversão, incluindo bloqueios primitivos e compostos. Estas versões finais são apresentadas na Tabela 16 e na Tabela 17, respectivamente.

A compatibilidade entre dois tipos de bloqueio compostos é derivada da compatibilidade de seus constituintes dois a dois. Em outras palavras, dois tipos de bloqueio compostos são compatíveis se, somente se, todos os pares resultantes do produto cartesiano de seus tipos de bloqueio constituintes forem pares de tipos compatíveis. Por exemplo, *rRpiR* e *iRpiW* são compatíveis porque (*rR**,** iR*), (*rR**,** piW*), (*piR**,** iR*) e (*piR**,** piW*) são todos pares compatíveis. Já *rRpiR* e *iRprW* são incompatíveis, pois (*rR**,** iR*), (*piR**,** iR*), (*piR**,** prW*) são pares compatíveis, mas o par (*rR**,** prW*) não é compatível.

Quanto à conversão envolvendo tipos de bloqueio compostos, a ideia é a mesma da conversão entre tipos primitivos. Ou seja, o tipo de bloqueio resultante tem que ser forte o suficiente para cobrir o tipo de bloqueio atual e o tipo de bloqueio requisitado. Novamente, os pares resultantes do produto cartesiano entre os tipos constituintes devem ser analisados, com base na Tabela 14, e o tipo resultante deve ser forte o suficiente para cobrir todos os pares. Por exemplo, suponha que o tipo de bloqueio atual seja *iRprR* e seja requisitado o tipo *rRpiR*, o novo tipo resultante da conversão é *riR* porque fazendo a conversão dos pares (*iR**,** rR*), (*iR**,** piR*), (*prR**,** rR*), (*prR**,** piR*), temos, respectivamente, (*riR*), (*iR*), (*rR*), (*priR*). Continuando a conversão dois a dois, da esquerda para direita, temos (*riR*), (*rR*) e (*priR*), depois, (*riR*) e (*priR*) e, por fim, (*riR*).

Além destas duas matrizes, o *Lock Manager* faz uso de uma tabela que conversão de bloqueio real para bloqueio planejado (ou de intenção). De acordo com a regra 8 do protocolo de aquisição/liberação de bloqueios, uma transação não pode liberar um bloqueio planejado em um item *x*, se a transação possuir correntemente um bloqueio em pelo menos um filho de *x*. Portanto, ao liberar um bloqueio real em um item que possua filhos com bloqueio, não podemos simplesmente liberar o bloqueio, temos que convertê-lo em um bloqueio planejado que será liberado mais tarde. Ocorre, portanto, uma espécie de *downgrade* de bloqueio real para correspondente bloqueio planejado, de acordo com Tabela 15. No caso de tipos de bloqueio compostos, o correspondente bloqueio planejado é obtido pela combinação, de acordo com a Tabela 14, dos bloqueios planejados dos constituintes. Por exemplo, o bloqueio real *rRpiR* gera o bloqueio planejado *priR*, pois constituintes (*rR*) e (*piR*) geram (*prR*) e (*piR*), respectivamente, que combinados geram (*priR*).

Tabela 15 - Conversão (downgrade) de bloqueio real em bloqueio planejado

Real Lock	***Planned Lock***
rR	prR
iR	piR
riR	priR
rW	prW
iW	piW
riW	priW
prR	prR
piR	piR
priR	priR
prW	prW
piW	piW
priW	priW
rRpiR	priR
rRprW	prW
rRpiW	piW
rRpriW	priW
iRprR	priR
iRprW	prW
iRpiW	piW
iRpriW	priW
riRprW	prW
riRpiW	piW
riRpriW	priW
rWpiW	priW
iWprW	priW

Tabela 16 - Matriz de compatibilidade de bloqueios completa

	rR	iR	riR	rW	iW	riW	prR	piR	priR	prW	piW	priW	rRpiR	rRprW	rRpiW	rRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
rR	s	s	s	n	s	n	s	s	s	n	s	n	s	n	s	n	s	n	s	n	n	s	n	n	n
iR	s	s	s	s	n	n	s	s	s	s	n	n	s	s	n	n	s	s	n	n	s	n	n	n	n
riR	s	s	s	n	n	n	s	s	s	n	n	n	s	n	n	n	s	n	n	n	n	n	n	n	n
rW	n	s	n	n	n	n	n	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
iW	s	n	n	n	n	n	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
riW	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
prR	s	s	s	n	s	n	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	n	s
piR	s	s	s	s	n	n	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	n
priR	s	s	s	n	n	n	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	n	n
prW	n	s	n	n	n	n	s	s	s	s	s	s	n	n	n	n	s	s	s	s	n	n	n	n	n
piW	s	n	n	n	n	n	s	s	s	s	s	s	s	s	s	s	n	n	n	n	n	n	n	n	n
priW	n	n	n	n	n	n	s	s	s	s	s	s	n	n	n	n	n	n	n	n	n	n	n	n	n
rRpiR	s	s	s	n	n	n	s	s	s	n	s	n	s	n	s	n	s	n	s	n	n	s	n	n	n
rRprW	n	s	n	n	n	n	s	s	s	n	s	n	n	n	n	n	s	n	s	n	n	n	n	n	n
rRpiW	s	n	n	n	n	n	s	s	s	n	s	n	s	n	s	n	n	n	n	n	n	n	n	n	n
rRpriW	n	n	n	n	n	n	s	s	s	n	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n
iRprR	s	s	s	n	n	n	s	s	s	s	n	n	s	s	n	n	s	s	n	n	s	n	n	n	n
iRprW	n	s	n	n	n	n	s	s	s	s	n	n	n	n	n	n	s	s	n	n	n	n	n	n	n
iRpiW	s	n	n	n	n	n	s	s	s	s	n	n	s	s	n	n	n	n	n	n	n	n	n	n	n
iRpriW	n	n	n	n	n	n	s	s	s	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
riRprW	n	s	n	n	n	n	s	s	s	n	n	n	n	n	n	n	s	n	n	n	n	n	n	n	n
riRpiW	s	n	n	n	n	n	s	s	s	n	n	n	s	n	n	n	n	n	n	n	n	n	n	n	n
riRpriW	n	n	n	n	n	n	s	s	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
rWpiW	n	n	n	n	n	n	n	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
iWprW	n	n	n	n	n	n	s	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Tabela 17 - Matriz de conversão de bloqueios completa

		Requested Lock Type																								
		rR	iR	riR	rW	iW	riW	prR	piR	priR	prW	piW	priW	rRpiR	rRprW	rRpiW	rRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
Old Lock Type	rR	rR	riR	riR	rW	iW	riW	rR	rRpiR	rRpiR	rRprW	rRpiW	rRpriW	rRpiR	rRprW	rRpiW	rRpriW	riR	riRprW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	iR	riR	iR	riR	rW	iW	riW	iRprR	iR	iRprR	iRprW	iRpiW	iRpriW	riR	riRprW	riRpiW	riRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	riR	riR	riR	riR	rW	iW	riW	riR	riR	riR	riRprW	riRpiW	riRpriW	riR	riRprW	riRpiW	riRpriW	riR	riRprW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	rW	rW	rW	rW	rW	riW	riW	rW	rW	rW	rW	rWpiW	rWpiW	rW	rW	rWpiW	rWpiW	rW	rW	rWpiW	rWpiW	rW	rWpiW	rWpiW	rWpiW	riW
	iW	iW	iW	iW	riW	iW	riW	iW	iW	iW	iWprW	iW	iWprW	iW	iWprW	iW	iWprW	iW	iWprW	iW	iWprW	iWprW	iW	iWprW	riW	iWprW
	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW	riW
	prR	rR	iRprR	riR	rW	iW	riW	prR	priR	priR	prW	piW	priW	rRpiR	rRprW	rRpiW	rRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	piR	rRpiR	iR	riR	rW	iW	riW	priR	piR	priR	prW	piW	priW	rRpiR	rRprW	rRpiW	rRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	priR	rRpiR	iRprR	riR	rW	iW	riW	priR	priR	priR	prW	piW	priW	rRpiR	rRprW	rRpiW	rRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	prW	rRprW	iRprW	riRprW	rW	iWprW	riW	prW	prW	prW	prW	priW	priW	rRprW	rRprW	rRpriW	rRpriW	iRprW	iRprW	iRpriW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	piW	rRpiW	iRpiW	riRpiW	rWpiW	iW	riW	piW	piW	piW	priW	piW	priW	rRpiW	rRpriW	rRpiW	rRpriW	iRpiW	iRpriW	iRpiW	iRpriW	riRpriW	riRpiW	riRpriW	rWpiW	iWprW
	priW	rRpriW	iRpriW	riRpriW	rWpiW	iWprW	riW	priW	priW	priW	priW	priW	priW	rRpriW	rRpriW	rRpriW	rRpriW	iRpriW	iRpriW	iRpriW	iRpriW	riRpriW	riRpiW	riRpriW	rWpiW	iWprW
	rRpiR	rRpiR	riR	riR	rW	iW	riW	rRpiR	rRpiR	rRpiR	rRprW	rRpiW	rRpriW	rRpiR	rRprW	rRpiW	rRpriW	riR	riRprW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	rRprW	rRprW	riRprW	riRprW	rW	iWprW	riW	rRprW	rRprW	rRprW	rRprW	rRpriW	rRpriW	rRprW	rRprW	rRpriW	rRpriW	riRprW	riRprW	riRpriW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	rRpiW	rRpiW	riRpiW	riRpiW	rWpiW	iW	riW	rRpiW	rRpiW	rRpiW	rRpriW	rRpiW	rRpriW	rRpiW	rRpriW	rRpiW	rRpriW	riRpiW	riRprW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	rRpriW	rRpriW	riRpriW	riRpriW	rWpiW	iWprW	riW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	rRpriW	riRpriW	riRprW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	iRprR	riR	iRprR	riR	rW	iW	riW	iRprR	iRprR	iRprR	iRprW	iRpiW	iRpriW	riR	riRprW	riRpiW	riRpriW	iRprR	iRprW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	iRprW	riRprW	iRprW	riRprW	rW	iWprW	riW	iRprW	iRprW	iRprW	iRprW	iRpriW	iRpriW	riRprW	riRprW	riRpriW	riRpriW	iRprW	iRprW	iRpriW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	iRpiW	riRpiW	iRpiW	riRpiW	rWpiW	iW	riW	iRpiW	iRpiW	iRpiW	iRpriW	iRpiW	iRpriW	riRpiW	riRpriW	riRpiW	riRpriW	iRpiW	iRpriW	iRpiW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	iRpriW	riRpriW	iRpriW	riRpriW	rWpiW	iWprW	riW	iRpriW	iRpriW	iRpriW	iRpriW	iRpriW	iRpriW	riRpriW	riRpriW	riRpriW	riRpriW	iRpriW	iRpriW	iRpriW	iRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	riRprW	riRprW	riRprW	riRprW	rW	iWprW	riW	riRprW	riRprW	riRprW	riRprW	riRpriW	riRpriW	riRprW	riRprW	riRpriW	riRpriW	riRprW	riRprW	riRpriW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	riRpiW	riRpiW	riRpiW	riRpiW	rWpiW	iW	riW	riRpiW	riRpiW	riRpiW	riRpriW	riRpiW	riRpriW	riRpiW	riRpriW	riRpiW	riRpriW	riRpiW	riRpriW	riRpiW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	riRpriW	riRpriW	riRpriW	riRpriW	rWpiW	iWprW	riW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRpriW	riRprW	riRpiW	riRpriW	rWpiW	iWprW
	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	riW	riW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	rWpiW	riW
	iWprW	iWprW	iWprW	iWprW	riW	iWprW	riW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	iWprW	riW	iWprW

3.6. Bloqueio Otimista

Apesar do objetivo deste trabalho ser o protocolo pessimista e os novos tipos de bloqueio apresentados anteriormente, esta seção expõe uma discussão muito interessante de como uma aplicação poderia aplicar a *abordagem* otimista, sob o ponto de vista das novas ideias apresentadas neste capítulo.

Com vimos na seção 2.1.2.2 (Bloqueio Otimista), o bloqueio otimista é uma aborgadem de detecção de conflitos. Portanto, o bloqueio (bloqueio virtual, na verdade não existe bloqueio físico) é adquirido *depois* que o usuário terminar o seu trabalho. O objetivo não é evitar conflitos, mas sim detectar conflitos ao final da transação, antes de persistir os dados (durabilidade). O usuário não tem garantia que *depois* que realizar seu trabalho e confirmá-lo (*commit*), o efeito resultante produzido será confirmado (durabilidade). Existe o risco de ter que "jogar fora" o trabalho, ao final, caso seja detectado um conflito.

Recordando, há duas formas de uma aplicação implementar o bloqueio otimista: acrescentar uma versão ao dado e verificar esta versão antes de modificar o dado ou uma forma mais geral descrita a seguir:

1. **Fase de Leitura**

   Carregamento dos dados (*loading*), ou seja, definição do RS (*Read Set*). Nesta fase, são realizadas as consultas (*queries*) que carregam o $RS_{initial}$. É preciso armazenar, na sessão do usuário, durante toda a transação, o $RS_{initial}$ e o código das consultas executadas.

2. **Fase de Validação (*pré-commit*)**

   Obter o bloqueio otimista, indicando que está tudo OK e pode persistir os dados. Obter bloqueio otimista significa simplesmente refazer as consultas armazenadas na fase de leitura, obtendo um novo RS, o RS_{commit}, e comparar com o $RS_{initial}$. Se o resultado da comparação for OK, o bloqueio otimista foi adquirido com sucesso, caso contrário, houve conflito e não foi possível adquirir o bloqueio otimista e, em geral, a transação tem que abortar.

   Existem dois cenários possíveis:

a. A aplicação somente faz uso da *abordagem* otimista
   Neste caso, basta todas as transações adquirirem seus bloqueios otimistas.
b. A aplicação faz uso de ambas as abordagens: otimista e pessimista
   Neste caso, todas transações devem, *em primeiro lugar*, adquirir um bloqueio pessimista, para remoção ou inserção, nos grânulos "*Property of Resource*" mencionados em todos os *statements* a serem removidos ou inseridos, antes de persisti-los, ou seja, detectar conflitos com transações concorrentes pessimistas. Por exemplo, antes de remover o *statement* `ex:mark foaf:name 'mark douglas'`, tem que obter, no *Lock Manager*, um bloqueio de escrita para remoção (rW) no grânulo "`property foaf:name of resource ex:mark`". Se conseguir adquirir todos os bloqueios pessimistas, ai sim tenta-se a aquisição dos bloqueios otimistas como é de praxe, ou seja, detectar conflitos com transações concorrentes otimistas.

3. **Fase de Escrita (*commit*)**
   Se conseguir obter o bloqueio otimista na fase anterior, a área de trabalho privada da transação é efetivamente persistida no banco de dados, concluindo o *commit*.

As fases 2 (validação) e 3 (escrita) têm que ocorrer atomicamente.

3.6.1. Bloqueios de Leitura para Inserção e Remoção

No controle otimista de transações RDF, conforme explicado na Tabela 18, também pode ser usada a noção de bloqueio de leitura para inserção e remoção. O tipo de bloqueio apenas define como será feita a validação.

Tabela 18 - Bloqueios otimistas de leitura para inserçã e remoção

	Optimistic Lock Obtido para verificar que "algo" não mudou
Removal	Obtido quando se quer *apenas* que não tenham ocorrido remoções de *statements*, por parte de transações concorrentes. Como obter este bloqueio otimista (como verificar): $\mathbf{RS_{initial} - RS_{commit} = \varnothing}$ Ou seja, ao refazer a consulta na fase de validação não pode haver *statements* que estejam presentes no *ResultSet* da consulta original, e não estejam presentes no *ResultSet* da consulta atual.
Insertion	Obtido quando se quer *apenas* que não tenham ocorrido inserções de *statements*, por parte de transações concorrentes. Como obter este bloqueio otimista (como verificar): $\mathbf{RS_{commit} - RS_{initial} = \varnothing}$ Ou seja, ao refazer a consulta na fase de validação não pode haver *statements* que estejam presentes no *ResultSet* da consulta atual, e não estejam presentes no *ResultSet* da consulta original.
Removal / Insertion	Caso uma transação precise obter bloqueio para remoção e inscrição (porque fez alterações, por exemplo), basta adquirir os dois bloqueios: $\mathbf{RS_{initial} - RS_{commit} = \varnothing \ \ AND \ \ RS_{commit} - RS_{initial} = \varnothing}$ Ou seja, ao refazer a consulta na fase de validação ambos os *ResultSets,* da consulta atual e da consulta orignal, têm que conter os mesmos *statements*.

3.6.2. Grânulos

Com em relação ao grânulos propostos, na versão otimista, um grânulo significa a abrangência da consulta a ser executada. Em tese, qualquer grânulo pode ser utilizado. Desde o grafo inteiro até uma única tripla.

Considerando os quatro grânulos propostos por este trabalho, três são facilmente mapeáveis em queries SPARQL. São eles:

- ***Graph***
  A consulta SPARQL seria:

```
CONSTRUCT { ?s ?p ?o}
WHERE { ?s ?p ?o .}
```

- ***Property***
  Por exemplo, para a propriedade `dbpedia2:birthPlace`, a consulta SPARQL seria:

```
CONSTRUCT { ?s dbpedia2:birthPlace ?o}
WHERE { ?s dbpedia2:birthPlace ?o .}
```

- ***Property of Resource***
  Por exemplo, para a propriedade `dbpedia2:birthPlace` do recurso `:Roger_Federer`, a consulta SPARQL seria:

```
CONSTRUCT { :Roger_Federer dbpedia2:birthPlace ?o}
WHERE { :Roger_Federer dbpedia2:birthPlace ?o .}
```

Não obstante, no caso do grânulo "*Resource*", o mapeamento não é tão direto assim. Em RDF, um recurso não existe por si só. Para um recurso existir tem que ter sido feita alguma declaração sobre este recurso. Um recurso existe quando existe pelo menos um *statement* que mencione o recurso. Sendo assim, para fazer uma consulta de um recurso, é necessário decidir quais *statements* definem o recurso. Em outras palavras, qual subgrafo é considerado uma representação para do recurso.

Uma aproximação interessante para representação de um recurso, que pode ser reaproveitada para o grânulo "*Resource*", na versão otimista, é a proposta conhecida por CBD - *Concise Bounded Description* [Sticker, 2005], cuja definição é:

Dado um nó específico (o nó inicial) em um grafo RDF particular (o grafo de origem), o CBD de um recurso, denotado pelo nó inicial, é um subgrafo que pode ser identificado com o seguinte algoritmo:

1. Inclua no subgrafo todos os *statements* no grafo de origem, onde o sujeito do *statement* é o nó inicial;

2. Recursivamente, para todos os *statements* incluídos no subgrafo até então, que possuam um nó em branco como objeto, inclua no subgrafo, todos os *statements* no grafo de origem, onde o sujeito do *statement* é o nó em branco em questão e que ainda não tenham sido incluídos no subgrafo;
3. Recursivamente, para todos os *statements* incluídos no subgrafo até então, para todas as reificações[43] de cada *statement* no grafo de origem, inclua o CBD, iniciando a partir do nó `rdf:Statement` de cada reificação.

O resultado é um subgrafo onde os nós objeto são ou URIrefs, ou literais, ou nós em branco que não são sujeitos de nenhum *statement* no grafo de origem. Um exemplo é mostrado na Tabela 19. Maiores detalhes podem ser obtidos em [Sticker, 2005].

Tabela 19 - Exemplo de CBD - Concise Bounded Description [Sticker, 2005]

Grafo de Origem

```
<?xml version="1.0"?>

<rdf:RDF
   xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl  ="http://www.w3.org/2002/07/owl#"
   xmlns:dc   ="http://purl.org/dc/elements/1.1/"
   xmlns:dct  ="http://purl.org/dc/terms/"
   xmlns:xsd  ="http://www.w3.org/2001/XMLSchema#"
   xmlns:foaf ="http://xmlns.com/foaf/0.1/"
   xmlns:ex   ="http://example.com/">

   <rdf:Description rdf:about="http://example.com/aReallyGreatBook">
      <dc:title>A Really Great Book</dc:title>
      <dc:publisher>Examples-R-Us</dc:publisher>
      <dc:creator>
         <rdf:Description>
            <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
            <foaf:name>John Doe</foaf:name>
            <foaf:mbox>john@example.com</foaf:mbox>
            <foaf:img>
               <rdf:Description rdf:about="http://example.com/john.jpg">
                  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Image"/>
                  <dc:format>image/jpeg</dc:format>
                  <dc:extent>1234</dc:extent>
               </rdf:Description>
            </foaf:img>
            <foaf:phone rdf:resource="tel:+1-999-555-1234"/>
         </rdf:Description>
      </dc:creator>
```

[43] Reificação RDF - Promover um *statement* RDF a recurso para que possa ser sujeito (ou objeto) de outros *statements*.

```
      <dc:contributor>
         <rdf:Description>
            <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
            <foaf:name>Jane Doe</foaf:name>
         </rdf:Description>
      </dc:contributor>
      <dc:language>en</dc:language>
      <dc:format>application/pdf</dc:format>
      <dc:rights>Copyright (C) 2004 Examples-R-Us. All rights reserved.</dc:rights>
      <dct:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2004-01-19</dct:issued>
      <rdfs:seeAlso rdf:resource="http://example.com/anotherGreatBook"/>
   </rdf:Description>

   <rdf:Statement>
      <rdf:subject rdf:resource="http://example.com/aReallyGreatBook"/>
      <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/format"/>
      <rdf:object>application/pdf</rdf:object>
      <rdfs:isDefinedBy rdf:resource="http://example.com/book-formats.rdf"/>
   </rdf:Statement>

   <rdf:Statement>
      <rdf:subject rdf:resource="http://xmlns.com/foaf/0.1/Image"/>
      <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/format"/>
      <rdf:object>image/jpeg</rdf:object>
      <rdfs:isDefinedBy rdf:resource="http://example.com/image-formats.rdf"/>
   </rdf:Statement>

   <rdf:Description rdf:about="http://example.com/anotherGreatBook">
      <dc:title>Another Great Book</dc:title>
      <dc:publisher>Examples-R-Us</dc:publisher>
      <dc:creator>June Doe (june@example.com)</dc:creator>
      <dc:format>application/pdf</dc:format>
      <dc:language>en</dc:language>
      <dc:rights>Copyright (C) 2004 Examples-R-Us. All rights reserved.</dc:rights>
      <dct:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2004-05-03</dct:issued>
      <rdfs:seeAlso rdf:resource="http://example.com/aReallyGreatBook"/>
   </rdf:Description>

   <rdf:Description rdf:about="http://example.com/aBookCritic">
      <ex:likes rdf:resource="http://example.com/aReallyGreatBook"/>
      <ex:dislikes rdf:resource="http://example.com/anotherGreatBook"/>
   </rdf:Description>

   <rdf:Property rdf:about="http://xmlns.com/foaf/0.1/mbox">
      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
   </rdf:Property>

</rdf:RDF>
```

CBD do recurso `http://example.com/aReallyGreatBook`

```
<?xml version="1.0"?>

<rdf:RDF
   xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl  ="http://www.w3.org/2002/07/owl#"
   xmlns:dc   ="http://purl.org/dc/elements/1.1/"
   xmlns:dct  ="http://purl.org/dc/terms/"
   xmlns:xsd  ="http://www.w3.org/2001/XMLSchema#"
   xmlns:foaf ="http://xmlns.com/foaf/0.1/">

   <rdf:Description rdf:about="http://example.com/aReallyGreatBook">
      <dc:title>A Really Great Book</dc:title>
      <dc:publisher>Examples-R-Us</dc:publisher>
      <dc:creator>
         <rdf:Description>
```

```
            <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
            <foaf:name>John Doe</foaf:name>
            <foaf:mbox>john@example.com</foaf:mbox>
            <foaf:img rdf:resource="http://example.com/john.jpg"/>
            <foaf:phone rdf:resource="tel:+1-999-555-1234"/>
         </rdf:Description>
      </dc:creator>
      <dc:contributor>
         <rdf:Description>
            <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
            <foaf:name>Jane Doe</foaf:name>
         </rdf:Description>
      </dc:contributor>
      <dc:language>en</dc:language>
      <dc:format>application/pdf</dc:format>
      <dc:rights>Copyright (C) 2004 Examples-R-Us. All rights reserved.</dc:rights>
      <dct:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2004-01-19</dct:issued>
      <rdfs:seeAlso rdf:resource="http://example.com/anotherGreatBook"/>
   </rdf:Description>

   <rdf:Statement>
      <rdf:subject rdf:resource="http://example.com/aReallyGreatBook"/>
      <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/format"/>
      <rdf:object>application/pdf</rdf:object>
      <rdfs:isDefinedBy rdf:resource="http://example.com/book-formats.rdf"/>
   </rdf:Statement>

</rdf:RDF>
```