Controle de Versões para Bancos de Dados Geográficos

Tiago Garcia de Senna Carneiro
Fernando Gibotti
Gilberto Ribeiro Queiroz
Instituto Nacional de Pesquisas Espaciais
{tiago, gibotti, gribeiro}@dpi.inpe.br

João Argemiro de Carvalho Paiva
Gilberto Câmara
Instituto Nacional de Pesquisas Espaciais
joao.paiva@oracle.com
gilberto@dpi.inpe.br

Resumo

O compartilhamento de informações geográficas em ambientes corporativos impõe regras de gerenciamento a fim de se garantir a integridade da base de dados. Vários usuários podem simultaneamente estar editando uma mesma informação, e no final das operações a base de dados deve incorporar as alterações de forma consistente. Para isso, técnicas de bloqueio de dados ou de controle de versões podem ser utilizadas em gerenciadores de banco de dados. Este trabalho propõe um esquema de controle de versão para o modelo de banco de dados da biblioteca Terralib, com o objetivo de permitir o acesso concorrente aos dados sem a necessidade de bloquear informações.

Abstract

On a shared environment with geographic information, there is a need for rules to guarantee the data integrity after concurrent access. Several users can edit the same data at the same time, and it is expected at the end of the processes that the information is stored consistently. For this situation, there are database techniques such as lock based or versioning control. This paper proposes a version control schema for Terralib library, with the goal to allow concurrent access without locking the shared data.

1. Introdução

Muitas aplicações geográficas demandam acesso concorrente às informações mantidas no banco de dados. Por exemplo, a digitalização do mapa topográfico de uma área extensa pode ser divido entre vários membros de uma equipe, onde cada membro deve digitalizar uma área do mapa e assim, um mesmo plano de informação (PI) do banco de dados geográfico (BDGeo) seria atualizado por vários usuários de forma simultânea. Portanto, um sistema de BDGeo deve oferecer mecanismos que garantam a consistência do banco quando vários usuários lêem e alteram os dados de forma concorrente.

Para controlar o acesso concorrente aos dados, os sistemas de gerenciamento de bancos de dados (SGBD) convencionais trabalham com o conceito de transações, onde transação leva o banco de dados de um estado consistente a outro consistente. Assim que uma transação é concluída, o estado anterior da base de dados é descartado. Uma maneira comum de se implementar transações num SGBD é através do bloqueio das tabelas envolvidas transação até que ela termine.

Nas aplicações geográficas uma operação de edição pode durar horas, dias ou meses, o que caracteriza uma transação de longa duração, para a qual não é adequado o bloqueio do DBGeo. Outras aplicações precisam manter a história do banco de dados, para que seja possível realizar estatísticas e análises sobre a evolução dos dados.

Conseqüentemente, existe a necessidade de se guardar mais de um estado da base de dados, afastando-se do modelo tradicional e incorporando-se do conceito de versão. Através do uso de versões, vários usuários podem editar simultaneamente dados geográficos, sem explicitamente bloquear os dados para outros usuários.

2. Conceitos básicos

Versões são cópias lógicas de algumas tabelas do DBGeo, não ocorrendo duplicação dos dados. Podem existir várias versões para um mesmo DBGeo, cada uma delas representa o banco em um determinado momento do tempo, ou sob um determinado ponto de vista, cujo registro é importante para a aplicação[5]. Uma versão não é afetada por mudanças em outras versões. No momento adequado, duas versões podem ser consistidas com relação a conflitos, dando origem a uma terceira versão que contém as propriedades das duas primeiras. Conflitos ocorrem quando um mesmo registro é diferente em duas versões sendo comparadas ou reconciliadas.

As versões devem ser organizadas em uma estrutura que reflita o relacionamento entre elas. A estrutura pode ser: (a) lista, onde cada versão só possui uma predecessora e uma sucessora, (b) árvore, onde várias versões podem ser derivadas de uma mesma versão, mas cada qual só tem uma predecessora (ORION [13] e Ode [8]) e (c) grafo acíclico dirigido, onde cada versão pode ter várias predecessoras e várias sucessoras (Iris [1] e AVANCE [2]).

No caso de uma lista, uma nova versão é sempre criada como sucessora da última versão, e representa a evolução do DBGeo no tempo. O relacionamento estabelecido entre as versões é um relacionamento de derivação. Quando a estrutura é uma árvore ou um grafo, se mais de uma versão é derivada da mesma versão, elas usualmente são chamadas de alternativas [7][9][10]. Versões que estão no mesmo caminho até a raiz da árvore ou grafo são chamadas de derivativas. A esta estrutura dá-se o nome de histórico de versões [9]. Quando o histórico de versões não é uma lista, o relacionamento de derivação também representa um relacionamento temporal, porém como uma ordenação parcial, pois só para as versões derivativas consegue-se determinar qual foi criada posteriormente.

Geralmente, as versões são classificadas para refletir o seu estado de desenvolvimento ou consistência [5]. Esta classificação se dá por meio da atribuição de um estado à versão. Por exemplo, a versão pode estar em um estado transitório onde ela ainda deve sofrer diversas alterações, antes de atingir um estado mais estável. Uma versão no estado de trabalho é uma versão que já atingiu um estágio mais estável, de tal forma que pode ser compartilhada, e conseqüentemente, não pode mais sofrer alterações, podendo, no entanto, ser removida. Uma versão no estado liberado é uma versão que chegou ao seu estágio final e, portanto, não pode mais ser alterada nem removida.

As versões podem ser promovidas ou rebaixadas pelo usuário ou de forma automática. Contudo, versões liberadas não podem ser rebaixadas. Um SGBD que permita controle de versões deve oferecer primitivas para consultar o histórico de versões e as próprias versões [5], tais como: (a) operações que buscam informações sobre uma versão específica: tempo, número ou nome associado a uma versão; (b) operações sobre versões relativas a uma versão específica: todas as predecessoras ou sucessoras; predecessora ou sucessora imediata, com base no tempo; predecessoras ou sucessoras com base no nome; usuários que derivaram a partir desta versão; e (c) operações sobre o grafo de versões como um todo: primeira, última, todas as versões, possivelmente restritas pelo nome e/ou tempo.

3. TerraLib

TerraLib [3] é uma biblioteca de código aberto para suportar o desenvolvimento de aplicações inovadoras em Geoprocessamento. Esta biblioteca oferece interfaces de programação para integração de todos os tipos de dados geográficos (sócio-econômicos, cadastrais, ambientais e imagens) em sistemas gerenciadores de banco de dados objeto-relacionais, assim como para o desenvolvimento de procedimentos de análise espacial [12]. A implementação desta biblioteca está baseada no conceito de múltiplos paradigmas de programação [4], onde se inclui o uso dos conceitos de programação genérica [6] e orientação a objetos [11].

3.1. Modelo de Dados TerraLib

A Figura 1 apresenta o modelo de dados simplificado do BDGeo implementado pela biblioteca TerraLib. Atributos chaves aparecem seguidos da nota (K) e atributos que são chaves estrangeiras são anotados com a simbologia (FK). Nesse modelo, a tabela te_layer representa um PI que é identificado unicamente pelo atributo layer_id. Cada PI pode ter uma ou mais representações para os dados geográficos nele contidos. Por exemplo, o mapa de solos de uma determinada região pode ter uma representação matrical (raster) ou poligonal. Os atributos repres_id, geom_table e geom_type da tabela te_representation permitem identificar as representações existentes para um PI, a tabela onde os dados daquela representação estão fisicamente armazenados e tipo de representação, respectivamente. Entre os tipos de representação possíveis encontram-se: matriz, pontos, linhas, polígonos, arco-nó, células e textos.

Um PI também pode estar associado a várias tabelas de atributos. A tabela te_layer_table armazena, para cada PI, o nome da tabela de atributos a ele associado (attr_table) e a coluna nesta tabela que permite associa-la às geometrias contidas no PI. Um PI pode ser associado com tabelas externas ao BDGeo através da tabela te_relation. Os atributos table_id, attr_link da tabela te_relation indicam qual tabela de atributos e qual das suas coluna será associada à tabela externa cujo nome é armazenado no atributo related_table através da coluna cujo nome é armazenado em related_link.

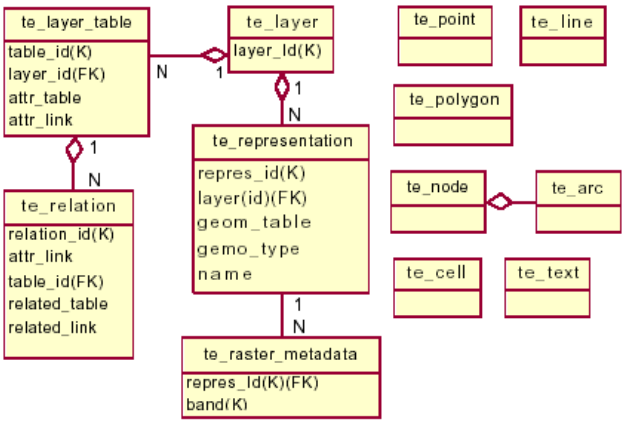

Figura 1. Modelo de dados simplificado da TerraLib.

4. Banco de dados versionados em TerraLib

Em geral um BDGeo é construído de forma incremental, e com o tempo novos PIs são adicionados ao banco para atender novas aplicações. Além disto, os dados contidos nos PIs podem ser atualizados ao longo do tempo. Por exemplo, geometrias podem ser alteradas, inseridas ou removidas assim como os atributos associados a objetos geográficos. Por conseguinte, o

sistema de controle de versão para um DBGeo deve ser capaz de manter o histórico dessas evoluções.

Frente a esses requisitos, três alternativas foram analisadas para a implementação do sistema de controle de versão para TerraLib (Figura 2): (a) versionamento total do DBGeo, onde seria possível remover, alterar e adicionar PIs durante a evolução do banco; (b) versionameto total do DBGeo, onde somente é permitida a adição de novos PIs; (c) versionamento individual de cada PI, permitindo a remoção e adição de PIs.

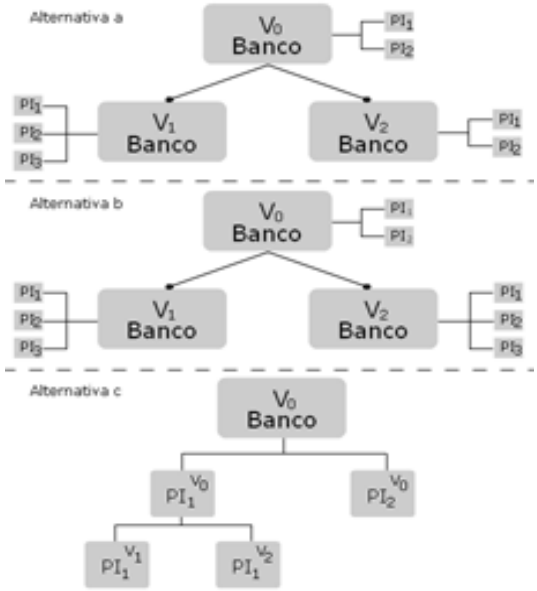

Figura 2. Alternativas para a implementação do sistema de controle de versão para TerraLib.

A alternativa (a) apresenta como principal desvantagem a grande complexidade de implementação. Devido ao fato de PIs existentes em versões anteriores poderem ser removidos nas versões derivadas, tanto a complexidade dos algoritmos utilizados para realizar a consistência entre as versões, quanto a demanda por processamento seriam aumentadas. Todos os temas, legendas e metadados associados ao PI removido deveriam ser atualizados. Seria preciso verificar se as tabelas de atributos que se relacionam com o PI removido possuem relacionamento com outros Pis. Caso não se relacionem, sua permanência no banco deveria ser analisada. Na alternativa (b) a complexidade de implementação é reduzida pelo impedimento de remoção de PIs ou da atualização de suas propriedades. Caso houvesse a necessidade de remover um PI do BDGeo e ainda assim manter uma versão do banco inalterada, seria preciso replicar todo o banco e, no banco replicado, suprimir o PI indesejado. A alternativa (c) é a mais flexível dentre as analisadas. Nela não ocorrre o versionamento total do DBGeo, e versões são criadas de forma individual para cada PI. A qualquer momento, é possível inserir um novo PI no banco de dados ou remover um PI existente que não possua versões derivadas. Sempre que um novo PI é adicionado ao banco, ele é rotulado como a versão zero daquele PI. Esta alternativa conserva a flexibilidade da alternativa (a), porém, reduz drasticamente sua complexidade de implementação e sua demanda por processamento.

As várias versões de um PI são armazenadas em uma estrutura de árvore, que permite recuperar o relacionamento derivativo entre as diversas versões e a criação de versões alternativas para um mesmo PI.

Durante o tempo de vida de uma versão, ela pode assumir dois diferentes estados: (a) aberta – a versão pode ser modificada ou removida do banco, mas não podem ser criadas versões derivadas a partir dela; (b) fechada – a versão não pode ser modificada nem removida do banco. Porém, novas versões podem ser derivadas a partir dela.

Duas versões podem ser comparadas ou reconciliadas. Quando ocorre um conflito durante essas operações, este é resolvido pelo usuário de maneira interativa.

Além de classificar o estágio de maturidade de uma versão, um estado também é o repositório de alterações realizados sobre o PI. Os estados são organizados em uma estrutura de árvore e são referenciados pelas versões. Cada versão referencia somente um estado, mas um estado pode ser referenciado por várias versões, com ilustra a Figura 3. O relacionamento entre versões e estados é melhor explicado através do cenário apresentado a seguir.

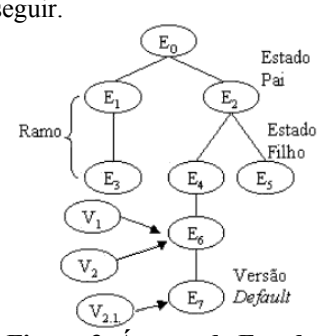

Figura 3. Árvore de Estados

Em um ambiente onde dois usuários acessam o DBGeo simultaneamente, o primeiro cria uma nova versão para um PI, a versão V1 e a ela adiciona dois objetos obj1 e obj2. Em seguida, ele reconcilia a versão V1 com sua predecessora, não encontra conflitos e a apaga. As Figuras 4 e 5 ilustram os efeitos dessas ações sobre o banco.

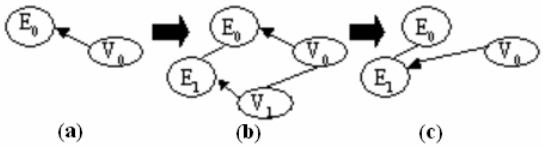

Figura 4. Alterações realizadas pelo primeiro usuário: (a) antes da alteração; (b) após a inserção dos objetos na versão V1; (c) após as versões serem reconciliadas.

Versão	Estado
V_0	1

OID	Estado	Dado
obj1	1	a
obj2	1	b

Figura 5. Tabelas de metadados e de objetos do PI após as versões serem reconciliadas.

As Figuras 6 e 7 ilustram os efeitos causados quando o segundo usuário cria uma versão V2, a ela adiciona o objeto obj3 e altera o objeto obj2.

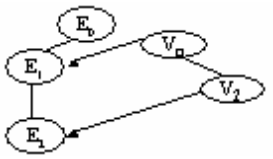

Figura 6. Efeitos das alterações realizadas pelo segundo usuário.

Versão	Estado
V_1	1
V_2	2

Metadados

OID	Estado	Dado
obj1	1	a
obj2	2	Bb
obj3	2	c

V_2

OID	Estado	Dado
obj1	1	a
obj2	1	b

V_0 (sem mudanças)

Figura 7. Tabelas de metadados e de objetos para as versões V0 e V2 do PI após alterações do segundo usuário.

Enquanto o segundo usuário permanece editando a versão V2, o primeiro cria uma nova versão V1, a ela adiciona o objeto obj4 e altera o objeto obj1. Como não há conflito entre as versões V0 e V1 ele as reconcilia, sem encontrar problemas. Os resultados dessas ações são apresentados pela Figuras 8 e 9.

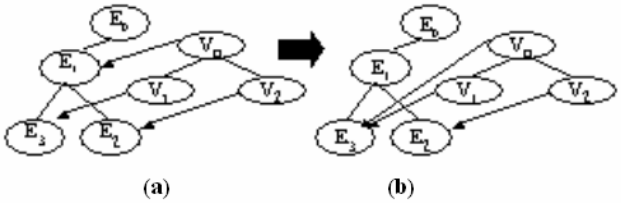

Figura 8. Efeito das alterações realizadas pelo primeiro usuário: (a) após a inserção e modificação dos objetos na versão V1; (b) após as versões serem reconciliadas e antes da versão V1 ser removida.

Versão	Estado
V_0	3
V_2	2
V_1	3

Metadados

OID	Estado	Dado
obj1	1	a
obj2	2	Bb
obj3	2	c

V_2 (sem mudanças)

OID	Estado	Dado
obj1	3	aa
obj2	1	b
obj4	3	d

V_0

OID	Estado	Dado
obj1	3	aa
obj2	1	b
obj4	3	d

V_1

Figura 9. Tabelas de metadados e de objetos para as versões V0, V1 e V2 do PI após as alterações do primeiro usuário.

Depois da versão V1 que estava aberta ser removida no final do processo de reconciliação, o primeiro usuário cria uma nova versão V1, altera o objeto obj2 , remove o objeto obj1 e então reconcilia esta versão com sua predecessora. As Figuras 10 e 11 mostram estas ações.

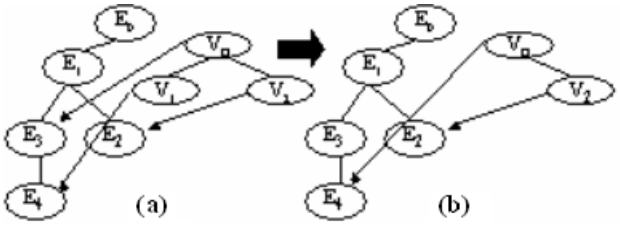

Figura 10. Alterações realizadas pelo primeiro usuário: (a) após a inserção e modificação dos objetos na versão V1; (b) após as versões serem reconciliadas.

Versão	Estado
V_0	4
V_2	2
V_1	4

Metadados

OID	Estado	Dado
obj1	1	a
obj2	2	Bb
obj3	2	c

V_2 (sem mudanças)

OID	Estado	Dado
obj1	4	<delete>
obj2	4	bbb
obj4	3	d

V_0

OID	Estado	Dado
obj1	4	<delete>
obj2	4	bbb
obj4	3	d

V_1

Figura 11. Tabelas de metadados e de objetos para as versões V0, V1 e V2 do PI após as alterações do primeiro usuário.

Então, o segundo usuário termina a edição de V2 e a reconcilia com a versão V0. Nesse processo, ele encontra alguns conflitos, resolve aceitar a alteração no objeto obj2 de V0 e aceita o objeto obj1 de V2. As Figuras 12 e 13 ilustram os efeitos dessas ações.

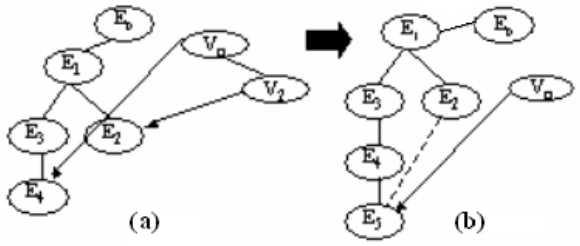

Figura 12. Efeito das alterações realizadas pelo segundo usuário: (a) antes da reconciliação; (b) após as versões serem reconciliadas.

Versão	Estado
V_0	5
V_2	5
V_1	4

Metadados

OID	Estado	Dado
obj1	5	a
obj2	4	bbb
obj3	2	c
obj4	3	d

V_2 (reconciliada)

OID	Estado	Dado
obj1	5	a
obj2	4	bbb
obj3	2	c
obj4	3	d

V_0 (reconciliada)

OID	Estado	Dado
obj1	4	<delete>
obj2	4	bbb
obj4	3	d

V_1 (sem mudanças)

Figura 13. Tabelas de metadados e de objetos para as versões *V0*, *V1 e V2* do PI após as alterações do segundo usuário.

No esquema de controle de versão, antes de o usuário consultar o BDGeo ele precisa informar sobre qual versão ele deseja obter dados. As consultas do usuário não são realizadas diretamente sobre as tabelas onde os dados estão fisicamente armazenados, mas sobre uma visão do banco de dados que reconstrói o conteúdo da versão desejada. Portanto, conforme a árvore de estados cresce pior se torna o desempenho das consultas realizadas pelo usuário. As operações de poda e compressão da árvore de estado podem ser utilizada para melhorar o desempenho das consultas ao DBGeo versionado.

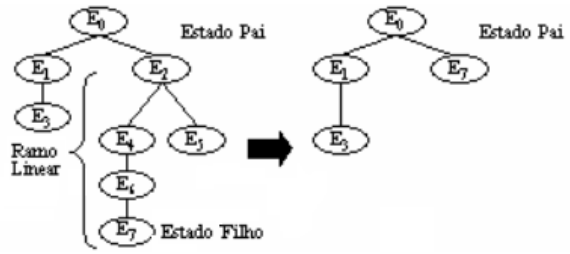

Figura 14. Processo de poda da árvore de estados.

O usuário pode comprimir a árvore de estados, removendo todos os estados que atualmente não são referenciados por uma versão. Também é possível podar a árvore de estados, reduzindo todos os ramos lineares da árvore: o estado filho irá substituir o estado pai do ramo e todos os estados que divergem do caminho direto serão descartados. As Figuras 14 e 15 ilustram o processo de poda e de compressão, respectivamente.

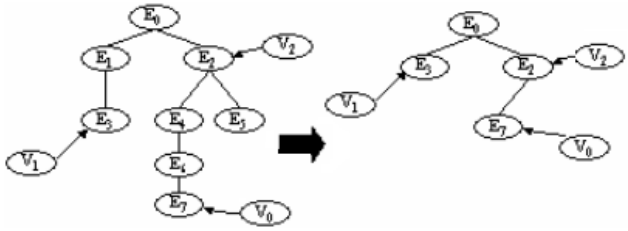

Figura 15. Processo de compressão árvore de estados.

4.1. Modelo de dados versionado para TerraLib

O modelo de dados apresentado na Figura 16 ilustra o fato de poderem existir várias versões de um mesmo PI pertencente a um DBGeo criado através da biblioteca TerraLib. A tabela te_version mantém informações sobre as versões de um PI, cada versão só pode pertencer a um único PI. Porém, um PI pode ter várias versões. Durante o tempo de vida de uma versão, ela pode assumir vários estados, mas em um dado instante do tempo, qualquer versão referencia apenas um estado. Contudo, um estado pode ser referenciado por várias versões simultaneamente.

Umas das principais diferenças entre o modelo de dados versionado para TerraLib e seu modelo de dados anterior, é o fato das tabelas te_layer_table e te_representation não mais se relacionarem diretamente com a tabela te_layer. Agora, estas duas tabelas passam a se relacionar com a agregação (te_layer + te_version). Por isso, a necessidade da adição do atributo version_id nessas tabelas. As tabelas te_version e te_state possuem o atributo parent_id que permite para um dado estado ou versão determinar a partir de qual versão ou estado este foi derivado. Desta maneira, armazena-se a estrutura de árvore na qual versões e estados são organizados.

Quando uma nova versão é criada, um novo estado associado a esta versão é criado, seu atributo initial_time é atualizado para a data em que esta ação ocorreu e seu atributo final_time recebe o valor nulo. Todos os registros presentes nas tabelas te_representation e te_layer_table associados à versão a partir da qual ela se originou são replicados. Os novos registros, frutos da replicação, têm o atributo version_id atualizado para o valor do identificador da versão que acaba de ser criada. Quando uma versão é fechada, ou removida seu atributo final_time é atualizado para a data em que a ação de fechamento ou remoção foi realizada. Se a versão foi removida, os registros das tabelas te_representation e te_layer_table a ela associados também são removidos.

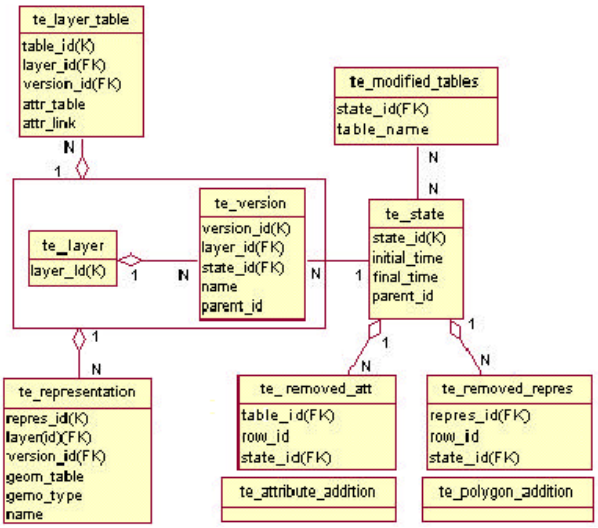

Figura 16. Modelo de dados versionado para TerraLib.

Para que não ocorra replicação de dados quando uma nova versão é criada, a estratégia adotada é armazenar somente registros das alterações realizadas sobre os dados, isto é, registros de atualizações, remoções ou adições. Para uma versão, sempre que uma de suas tabelas é alterada, o nome dessa tabela e o estado em que a alteração ocorreu são anotados na tabela te_modified_tables. Quando registros de uma representação ou das tabelas de atributos associados a uma determinada versão de um PI são removidos, o id da representação ou da tabela de atributos modificada e os números dos registros removidos são anotados nas tabelas te_removed_repres ou te_removed_attr, respectivamente.

Quando linhas de atributos ou de geometrias são adicionadas a alguma tabela, seja ela de atributos ou de geometrias, uma nova tabela com a mesma estrutura da tabela onde as linhas estão sendo adicionadas é criada e as novas linhas são inseridas nessa tabela. Por exemplo, na Figura 16, uma tabela te_polygon_addition, cuja estrutura é idêntica à estrutura da tabela te_polygon, seria criada para cada versão de um PI onde ocorreu inserção de novos polígonos. Da mesma maneira, quando novas linhas de atributos são inseridas na tabela de atributo associada a uma versão de um determinado PI, uma tabela com a mesma estrutura da tabela de atributo é criada e a nova linha é nela inserida. Na Figura 16, uma tabela te_attribute_addition é criada para cada tabela de atributo alterada numa determinada versão.

Para cada versão de um PI, existe uma visão criada no banco de dados. Esta visão remonta o conteúdo da versão a partir da tabelas bases, aquelas pertencentes à versão zero do PI, e a partir das tabelas de remoções (te_removed_repres e te_removed_attr) e adições (te_attribute_addition, te_polygon_addition, etc.). Nas consultas realizadas sobre o BDGeo, o usuário deixa de acessar diretamente as tabelas base onde os dados estão armazenados e passa a acessar a visão associada à versão sendo consultada.

5. Considerações Finais

Este trabalho propôs um esquema de controle de versões para o modelo de banco de dados da biblioteca Terralib. O modelo de controle de versão proposto ainda se encontra sob implementação, sendo necessário a realização de testes para analisar sua adequação como solução viável para o problema em questão e para avaliar seu desempenho quando utilizado em um ambiente multi-usuário e para grandes volumes de dados.

6. Referências

[1] D. Beech, B. Mahbod, Generalized Version Control in an Object-Oriented Database. In: International Conference on Data Engineering, Febr. 1-5, Los Angeles-EUA. Proceedings. IEEE. 1988. p.14-22.

[2] A. Björnerstedt,; C. Hultén, "Version Control in an Object-Oriented Architecture". In: KIM, W.; Lochovsky, F.H. (eds.). Object-Oriented Concepts, Databases, and Applications. Reading-Mass, Addison-Wesley, chap. 18, p. 451-485, 1989.

[3] G. Câmara, R. Souza, B. Pedrosa, L. Vinhas, A. Monteiro, J. Paiva, M. Carvalho, M. Gatass, "TerraLib: Technology in Support of GIS Inovation", Proceedings of II Workshop Brasileiro de Geoinformação, pp. 126-133, São Paulo, 2000.

[4] J. Coplien, *Multi-paradigm Design for C++*: Addison-Wesley, 1999.

[5] L. Golendziner; C. Santos, "Versions and configurations in object-oriented database systems: a uniform treatment". In: *Int. Conf. on Management of Data (COMAD)*, 7., 1995. p.18-37.

[6] M. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*: Addison-Wesley, 1999.

[7] M. Fauvet "Modelling and managing versions and histories in an object-oriented environment", Grenoble, IMAG/BULL, July 1991. (Rapport RAP015)

[8] R. Agrawal, S. Buroff; N. Gehani ; D. Shasha, "Object Versioning in Ode". In: Data Engineering, 7., April 8-12, 1991. Proceedings. IEEE, p. 446-455.

[9] R. Katz, "Toward a unified framework for modeling in engineering databases". ACM Computing Surveys, New York , v.22, n.4 , p. 375-408, .Dec 1990.

[10] S. Zdonik, "Version Management in an Object-Oriented Database". In: International Workshop on Advanced Programming Environments, June 1986, Trondheim-Noruega. Proceedings. Springer-Verlag, 1987. p.405-422.

[11] S. Zdonik, "Object-Oriented Type Evolution". In: Bancilhon, F; Buneman, P.(eds). Advances in Database Programming Languages. Addison-Wesley, p.277-288, 1990.

[12] Z. Zhang and D. Griffith, "Integrating GIS components and spatial statistical analysis in DBMSs," *International Journal of Geographical Information Science*, vol. 14, pp. 543-566, 2002.

[13] W. Kim, E. Bertino, J. Garza, "Composite objects revisited". In: ACM Sigmod Conference, May 31-June 2, 1989, Portland-Oregon. Proceedings. p.337-34.