UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

TVMSE - Uma Implementação do Versionamento de Esquemas segundo o Modelo TVM

por

ANELISE JANTSCH

Dissertação submetida à avaliação
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof[a]. Dr[a]. Nina Edelweiss
Orientadora
Prof. Dr. Clesio Saraiva Dos Santos
Co-orientador

Porto Alegre, maio de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Jantsch, Anelise
TVMSE – Uma Implementação do Versionamento de Esquemas segundo o Modelo TVM / por Anelise Jantsch. – Porto Alegre : PPGC da UFRGS, 2003.
97 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2003. Orientadora: Edelweiss, Nina; Co-Orientador: Santos, Clesio Saraiva dos.

1. Versionamento de Esquemas. 2. Evolução de Esquemas. 3. Banco: Dados temporais. 4. Banco: Dados Orientado a Objetos. 5. Modelo Temporal de Versões. I. Edelweiss, Nina. II. Santos, Clesio Saraiva dos. III. Título.

“A Vida não está contida na escala do tempo,

não está contida na escala da caducidade;

o tempo pelo contrário, está na palma das mãos da Vida,

a qual cerrada se torna um ponto e, aberta, se torna infinito.”

Masaharu Taniguchi

Agradecimentos

Em primeiro lugar quero agradecer a Deus pelo milagre da vida e por mais este objetivo alcançado.

Gostaria de agradecer à profesora Nina, minha orientadora, e ao professor Clesio, meu co-orientador, pela maravilhosa oportunidade de ter trabalhado com profissionais tão competentes e brilhantes. Obrigado pelo trabalho desenvolvido!

Agradeço a todos os colegas do PPGC por terem influenciado através de suas opiniões a realização deste objetivo. Em especial, aos colegas do grupo Versões-Tempo: Adriana, Aglaê, Carlos, Cláudio, Fábio, Fabrício, Lialda, Mirella, Paôla, Raquel, Renata e Silvia. A todos muito obrigada!

Não poderia deixar de agradecer ao bolsista de iniciação científica Lúcio Otávio Marchioro Rech, pela ajuda imprescindível na implementação do protótipo. Sentirei falta das nossas reuniões semanais e de responder as suas dúvidas "cabalísticas".

Agradeço a todos os professores e funcionários do Instituto de Informática.

Agradeço à FAURGS e ao CNPq pela bolsa concedida.

Não seria possível atingir esta meta sem o apoio constante da família e de amigos verdadeiros. Agradeço aos meus pais Arno e Maria, por terem me ensinado o valor do caráter e da honestidade. Sou muito feliz de ser filha de pessoas tão maravilhosas, vocês são e serão sempre os meus referenciais na vida. Agradeço a minha irmã Belquise, por ter sido compreensiva ao meu isolamento e aos meses de mau-humor. Mana, te admiro muito e te desejo sucesso em todos os momentos da tua vida!!!

Encerrado os agradecimentos aos familiares não poderia esquecer dos amigos, começando pelos mais antigos: Marcelo, Lilian e Cláudia tive muita sorte de conhecê-los na infância e continuarmos nossa amizade até os dias de hoje. Obrigada pelas viagens curtas, para que eu pudesse voltar a estudar. Afonso, amigo querido obrigada por tudo, por ter ficado muitas vezes em silêncio ao meu lado quando eu não queria falar, obrigada por me apoiar sempre e me acompanhar em todos os momentos. Ana Cecília e Carol, vocês são responsáveis por me alegrar sempre, por me tirarem de casa simplesmente para saber como eu estava. Margareth, obrigada por me ouvir e ajudar a superar os obstáculos que surgiram no ano passado. Não poderia esquecer da turma do RU da UFRGS, com vocês os almoços se tornaram momentos de descontração e apoio mútuo, juntos nos tornamos muito fortes, agradeço a Deise, Alessandra, Vânia, Everaldo, Diana, Luciana S., Luciana F., Guilherme, André, Delcino. Agradeço aos amigos do Pândega, em especial a Grace, Adriana, Henrique e Bia.

Gostaria de agradecer a todos que de alguma forma colaboraram com seu sorriso, suas palavras de incentivo e tornaram amena a caminhada. A todos muito obrigada!

Sumário

Lista de Abreviaturas

BD	Banco de Dados
BDOO	Banco de Dados Orientado a Objetos
BDT	Banco de Dados Temporal
DBA	Database Administrator
GAD	Grafo Acíclico Dirigido
MTV	Multiple Table Version
OO	Orientado a objetos
PMTV	Partial Multiple Table Version
SGBD	Sistema Gerenciador de Banco de Dados
SGBDT	Sistema Gerenciador de Banco de Dados Temporal
SIG	Sistemas de Informação Geográfica
STV	Single Table Version
TVM	Temporal Versions Model
TVMSE	Temporal Versions Model Schema Evolution
TVQL	Temporal Versions Query Language

Lista de Figuras

Lista de Tabelas

Resumo

Um esquema de banco de dados certamente sofrerá alguma alteração com o passar do tempo. Algumas das causas destas modificações são ocorrência de um aumento no domínio do sistema, erros ocorridos na fase de projeto, mudanças na realidade representada pelo sistema, ou a necessidade de melhoria no seu desempenho. O uso de bancos de dados temporais é uma alternativa para o armazenamento das informações da evolução, pois permite sua recuperação por meio do histórico de suas mudanças.

O presente trabalho propõe um ambiente para implementar evolução de esquemas sobre um BDOO, utilizando o Modelo Temporal de Versões (TVM). Deste modo, características de versões e de tempo são utilizadas tanto no nível dos esquemas como nos dados armazenados.

Estados são associados às versões de esquema para representar seus estágios de desenvolvimento durante a evolução.

O gerenciamento das versões de esquema é realizado por intermédio de uma camada denominada meta-esquema. Em um outro nível, o gerenciamento das instâncias é realizado por meio de uma camada denominada metadados, inserida para cada versão de esquema definida. Por intermédio destes controles é possível analisar a evolução dos esquemas como um todo e, para cada esquema, as correspondentes versões de seus dados e sua evolução temporal. Algumas alternativas de consulta para um ambiente com estas características são analisadas.

O trabalho apresenta, ainda, as características básicas de um protótipo implementado para verificar a viabilidade da proposta apresentada.

Palavras-chave: Versionamento de Esquemas, Evolução de Esquemas, Bancos de Dados Temporais, Bancos de Dados Orientados a Objetos, e TVM.

TITLE: “TVMSE - A SCHEMA VERSIONING IMPLEMENTATION UNDER TVM MODEL”

Abstract

A database schema certainly will suffer some modification during its lifetime. Several possibilities may cause these modifications as an increase in the system domain, the detection of project errors, changes in the modeled reality, or the requirement of performance improvement. Temporal databases are an alternative for the storage of the historic information, allowing their retrieval according to the evolution history.

The present work presents an environment to implement schema evolution on an OODB, using the Temporal Versions Model (TVM). Thus, version and time characteristics are available not only in the schema level, but also for stored data.

States are associated to the schema versions, representing the development stages during the evolution.

Schema version management is achieved through a layer named Meta-Schema. On another level, instance management is provided through a layer named Meta-Data, inserted for every schema version. These controls assure the analisys of the whole schema evolution, and for each schema version, the analisys of the correspondent data versions and their temporal evolution. Some query alternatives for an environment with these features are also analyzed.

This dissertation also presents the basic features of a prototype implemented to verify the viability of the proposal.

Keywords: Schema Versioning, Schema Evolution, Temporal Databases, Object Oriented Databases, and TVM.

1 Introdução

A comunidade científica tem pesquisado amplamente sobre o tópico banco de dados temporais, o qual tem toda uma terminologia e características próprias que serão abordadas mais adiante neste trabalho.

O conceito de tempo em bancos de dados está diretamente ligado ao conceito de evolução dos objetos na base. Este objetivo é alcançado por meio do histórico das alterações dos objetos no banco de dados. Assim, como os objetos sofrem alterações com o passar do tempo, também o esquema conceitual precisa ser adaptado para melhor refletir a realidade da aplicação que representa. Quando utilizados bancos de dados temporais, é importante a manutenção do histórico das mudanças do esquema durante sua evolução. Esta tarefa normalmente é de gerenciamento bastante complexo, o que aumenta ainda mais quando se trata de bancos de dados temporais (RODDICK et al., 2000; MANDREOLI, 2001).

O conceito de versão pode ser utilizado como técnica de representação e manipulação dos dados históricos armazenados durante a evolução do esquema. No campo da orientação a objetos, dois tipos de versões de esquema têm sido consideradas: alternativas e temporais. A primeira é típica de ambientes de projeto (aplicações CAD/CAM) e a segunda é necessária para modelar a história das mudanças estruturais (mais apropriada para aplicações SIG e multimídia).

Em Moro et al.(2001;2001-a) foi proposto um modelo que permite o versionamento de objetos e a manutenção do histórico destas versões – o Modelo Temporal de Versões (Temporal Versions Model – TVM). Em um trabalho anterior Moreira (1999); Moreira e Edelweiss (1999), foi realizado um estudo aprofundado acerca da evolução de esquemas em bancos de dados temporais, tendo sido uma alternativa implementada em Angonese (2000). Estes estudos foram utilizados como base para a realização deste trabalho.

O objetivo desta dissertação é a implementação de um protótipo que simule a evolução de esquemas sobre um banco de dados relacional utilizando o TVM como modelo de dados. A motivação deste trabalho está baseada na literatura com respeito aos aspectos temporais na evolução de esquemas. Com base nestes conceitos foi simulado, por intermédio de um protótipo, um ambiente de versionamento de esquemas, onde as versões de esquema são de tempo de transação e os dados associados são bitemporais, implementando o Modelo Temporal de Versões.

1.1 Organização do texto

O documento está organizado da seguinte maneira:

- o capítulo 2 apresenta uma revisão das principais características dos bancos de dados temporais e do conceito de versão;
- o capítulo 3 apresenta os aspectos necessários a um ambiente temporal com suporte ao versionamento de esquemas;
- o capítulo 4 apresenta um estudo dos trabalhos relacionados na área de evolução de esquemas;
- o capítulo 5 apresenta as principais características do TVM, bem como sua linguagem de consulta, TVQL;
- o capítulo 6 apresenta a análise do versionamento de esquemas, tendo o TVM como modelo de dados, utilizada na fundamentação da implementação do protótipo;
- o capítulo 7 apresenta o protótipo implementado;

- o capítulo 8 apresenta um estudo de caso;
- o capítulo 9 apresenta as conclusões.

2 Conceitos de tempo e versão

A habilidade de modelar a dimensão temporal do mundo real é essencial para um grande número de aplicações, tais como indicadores econômicos, transações bancárias, controle de inventário, registros médicos, sistemas de tempo real, multimídia, reservas de linhas aéreas, versões de aplicações em CAD/CAM, aplicações estatísticas e científicas. Sistemas gerenciadores de bancos de dados (SGBDs) que suportam estas aplicações precisam ser capazes de satisfazer características temporais (GORALWALLA, ÖZSU, SZAFRON, 1998).

Em contrapartida, o uso de versões Golendziner (1995); Galante (1998); Elmasri e Navathe (2000); Roma et al. (2001) permite o registro de diferentes alternativas de projeto e torna mais eficiente a recuperação de falhas quando uma nova versão leva a base de dados a um estado inconsistente. Além disso, possibilita o suporte ao trabalho corporativo e a manutenção de diferentes representações para uma mesma informação. No caso da evolução de esquemas, especificamente, versões podem ser utilizadas para a manutenção do histórico das modificações realizadas em bancos de dados, o que autoriza atualizações futuras tornando as opções de consulta ainda mais ricas.

Neste capítulo são apresentados os conceitos básicos de tempo e versão, necessários à compreensão do versionamento de esquemas sobre um banco de dados temporal.

2.1 Tempo

No mundo real, a maioria dos eventos está relacionada com o tempo, por meio de calendários, prazos, restrições e outros. Esta característica, entretanto, nem sempre é representada de maneira apropriada, ou nem é realizada de forma automática pelo SGBD.

Bancos de dados temporais são utilizados para uma representação apropriada dos aspectos de tempo, que é realizada mediante a utilização de rótulos temporais, também chamados de *timestamps*. Um rótulo temporal é uma informação temporal associada a um determinado valor, o que possibilita a manutenção do histórico das alterações que ocorrem sobre este valor através do tempo. Os rótulos temporais podem ser classificados de acordo com o tipo de tempo que representa como tempo de transação, tempo de validade e tempo definido pelo usuário (TANSEL, 1993; EDELWEISS, OLIVEIRA, 1994; ZANIOLO, 1997). O tempo de transação é o rótulo temporal fornecido pelo próprio SGBD, isto é, o instante em que a operação foi realizada no banco de dados. O tempo de validade representa o momento em que a informação armazenada é verdadeira na realidade modelada. Por fim, o tempo definido pelo usuário caracteriza as propriedades temporais definidas em um domínio temporal e manipuladas pelos programas de aplicação, sendo sua semântica definida e conhecida somente pela aplicação (ELMASRI, NAVATHE, 2000).

A seguir serão apresentados alguns conceitos básicos utilizados na especificação de aspectos temporais (MCKENZIE, SNODGRASS, 1991; TANSEL, 1993; JENSEN ET AL., 1998; ELMASRI, NAVATHE, 2000):

- *chronon* – é a menor duração de tempo suportada por um banco de dados temporal, não podendo ser decomposta;
- *lifespan* – é o tempo no qual o objeto é definido;
- *span* – é um período de tempo com duração conhecida, também chamado simplesmente de período;

- instante de tempo – a informação existe em um instante de tempo, também chamado de ponto no tempo;
- intervalo temporal – é o tempo compreendido entre dois instantes de tempo;
- elemento temporal – é um conjunto disjunto de intervalos temporais;
- tempo linear – implica em uma total ordenação entre quaisquer dois instantes no tempo;
- tempo ramificado – é permitido que dois instantes de tempo diferentes sejam sucessores ou antecessores imediatos de um mesmo instante;
- tempo circular – permite a modelagem de eventos e processos recorrentes;
- tempo contínuo – supõe-se que o tempo é contínuo por natureza;
- tempo discreto – baseia-se em uma linha de tempo composta de uma seqüência de intervalos temporais consecutivos (*chronons*), que não podem ser decompostos e possuem idêntica duração;
- granularidade temporal – é a duração de um *chronon* (minuto, dia, ano) que permite uma representação melhor da realidade. É determinada pela aplicação.

De acordo com a maneira utilizada para armazenar valores temporais, os bancos de dados podem ser classificados em quatro tipos diferentes: instantâneos, de tempo de transação, de tempo de validade e bitemporais (EDELWEISS, OLIVEIRA, 1994; JENSEN ET AL., 1998; ELMASRI, NAVATHE, 2000). Estas classificações estão apresentadas em maiores detalhes a seguir:

- Bancos de Dados Instantâneos – este tipo de banco de dados não suporta informações temporais de maneira implícita. É chamado de instantâneo porque a única informação existente é a presente, não existindo representação quanto às transições dos estados. A maioria dos sistemas gerenciadores de banco de dados (SGBDs) são deste tipo;
- Bancos de Dados de Tempo de Transação – também chamados de *rollback*, associam aos dados seu tempo de transação. Este tempo é fornecido de forma automática pelo SGBD. Os valores passados ficam disponíveis para serem consultados;
- Bancos de Dados de Tempo de Validade – também chamados de históricos, associam aos dados seu tempo de validade. Este tempo deve ser fornecido pelo usuário. O acesso à informação é realizado de acordo com os momentos em que esta é válida;
- Bancos de Dados Bitemporais – a cada informação são associados seus tempos de transação e de validade. É a forma mais completa de armazenar informações quanto ao aspecto temporal. É possível o acesso a toda a história do banco de dados, tanto pelas transações como pelas validades dos dados. Em bancos de dados bitemporais nenhum atributo sofre alteração, exceto aqueles cujo tempo de transação final está em aberto (ELMASRI, NAVATHE, 2000);
- Bancos de Dados Multitemporais (CASTRO, GRANDI, SCALAS, 1995, 1997; CASTRO, 1997) – são bancos de dados nos quais podem ser encontrados dados dos tipos instantâneos, de tempo de transação, de tempo de validade e bitemporais. A princípio não existem limitações de

quantas dimensões um banco de dados pode suportar, por isso bancos de dados multitemporais são possíveis de fato (SKJELLAUG, 1997).

Embora a maioria dos modelos temporais tenha sido projetada para suportar as necessidades de uma aplicação particular ou de um grupo de aplicações similares, se forem analisadas as funcionalidades oferecidas pelos modelos temporais podem ser identificadas características comuns, tais como (JENSEN ET AL., 1998):

- cada modelo temporal tem uma ou mais primitivas temporais (instante temporal, intervalo temporal, rótulo temporal). O domínio discreto ou contínuo é utilizado em cada modelo temporal como um domínio temporal sobre estas primitivas;
- alguns modelos temporais suportam primitivas temporais de mesma granularidade, enquanto outros permitem que as primitivas temporais possam ser especificadas em diferentes granularidades;
- a maioria dos modelos temporais permite um modelo de tempo linear, enquanto poucos aceitam um modelo de tempo ramificado. No primeiro, as primitivas temporais são totalmente ordenadas, enquanto no segundo elas possuem uma ordenação parcial definida;
- todos os modelos temporais oferecem algum tipo de modelagem da informação histórica sobre entidades do mundo real e/ou histórias das entidades da base de dados. Dois dos mais populares tipos de histórias que têm sido empregados são os históricos de transação e validade respectivamente.

Esta seção apresentou, de forma resumida, as principais características relacionadas a bancos de dados temporais. Cabe salientar que, apesar dos esforços em pesquisas e em implementação de protótipos, ainda não existe comercialmente um banco de dados temporal.

No capítulo 5 serão apresentadas as características temporais do TVM, onde muitos dos conceitos aqui abordados serão revisitados para uma melhor compreensão do versionamento de esquemas sobre o TVM.

2.2 Versão

Bancos de dados orientados a objetos (BDOO) permitem a modelagem de entidades do mundo real como objetos armazenados. Diversos domínios da aplicação têm requisitado que sejam mantidos não só um estado de uma entidade, mas vários, representando diferentes estágios da entidade em tempos distintos ou sob pontos de vista diversos. Este requisito é modelado por meio de versões de um objeto (GOLENDZINER, 1995).

Considerando bancos de dados orientados a objetos, pode-se resumir o uso de versões mediante os seguintes objetivos (GOLENDZINER, 1995):

- manter a história de um objeto. Em bancos de dados históricos, onde há a necessidade de realizar estatísticas ou análises sobre a evolução dos dados ou em aplicações de projeto, onde é necessário armazenar as várias tentativas realizadas pelos projetistas ou os estágios de evolução do projeto;
- tratar mudanças nas definições dos tipos (e/ou classes) do banco de dados. Em sistemas convencionais, tais mudanças exigem que todas as instâncias do tipo e todos os programas que dele dependem sejam adaptados para a nova definição. Versões de objeto podem ser utilizadas

para evitar constantes reorganizações do banco de dados e alterações nos programas;

- manter a confiabilidade dos dados em ambientes concorrentes. Nestes ambientes, transações são executadas concorrentemente e os problemas que devem ser enfrentados são o controle de concorrência, as transações que devem ser desfeitas e a restauração do banco de dados.

A utilização da técnica de versionamento apresenta alguns inconvenientes, como, por exemplo, o tempo adicional despendido na geração, manutenção e controle da proliferação das versões. A complexidade imposta pela navegação no grafo de herança tradicional é também aumentada pela necessidade de navegação em uma hierarquia de versões. Em relação à adaptação das instâncias presentes no banco de dados, a técnica de versões consiste em gerar versões de objetos para cada nova versão derivada de classe ou esquema na estrutura.

As versões de instâncias podem ser criadas livremente pelo usuário, desde que monitoradas pelo SGBD, ou podem ser geradas automaticamente por cópia ou conversão de uma versão de instância para outra, desde que seja utilizado o esquema de dados correspondente.

2.2.1 Relacionamento entre versões

Versões de um objeto podem estar representando a sua evolução no tempo, diferentes alternativas de modificações, ou, ainda, uma combinação de ambas. A possibilidade de representar alternativas de um objeto é especialmente importante para aplicações de projeto. Versões devem ser organizadas em uma estrutura que reflita o relacionamento existente entre elas. A estrutura pode ser uma das três seguintes:

1- lista, onde cada versão só tem uma versão antecessora e uma sucessora;
2- árvore, onde várias versões podem ser derivadas de uma mesma versão, mas cada uma delas só tem uma antecessora;
3- grafo acíclico dirigido, onde cada versão pode ter várias antecessoras e várias sucessoras.

No caso de uma lista, uma nova versão é sempre criada como sucessora da última e representa a evolução de um objeto no tempo. O relacionamento estabelecido entre as versões é de **derivação**. Quando a estrutura é uma árvore ou grafo, ela traz mais informações além da relação de derivação entre versões e é utilizada por aplicações de projeto, quando diferentes estratégias devem ser exploradas em paralelo. Se mais de uma versão é derivada de uma mesma elas usualmente são chamadas de **alternativas**, pois representam diferentes tentativas de solucionar um problema. Estes conceitos estão ilustrados na figura 2.1.

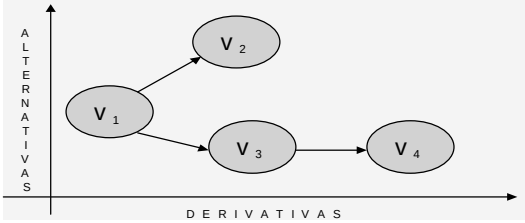


FIGURA 2.1 - Representação de versões

2.2.2 Derivação de versões

A criação de uma nova versão de esquema, classe e/ou método pode ser explícita, quando o controle é realizado pelo usuário, ou implícita, quando derivada automaticamente pelo sistema.

A derivação implícita é a operação normalmente utilizada para retratar a seqüência histórica de evolução do esquema. Assim, uma nova versão é derivada em decorrência de operações de modificação aplicadas sob o esquema ou para fins de adaptação das instâncias presentes no banco de dados.

A derivação explícita de versões é especificada pelo usuário e representa a cópia de uma (ou mais) versão(ões) antecessora(s) do grafo de derivação, permanecendo livre para modificações e testes. Nesse caso, todo controle deve ser realizado manualmente pelo usuário.

2.2.3 Modelo de Versões

O Modelo de Versões (GOLENDZINER, 1995), adota a técnica de versões alternativas, isto é, elas permanecem armazenadas em um espaço comum, evoluem em paralelo e operam sobre a mesma coleção de dados.

Uma versão é um objeto que possui seu próprio identificador de objeto (OID), permitindo que seja diretamente manipulada e consultada. As versões de uma entidade do mundo real ficam agrupadas e constituem um objeto versionado, o qual também possui um OID que mantém informações sobre as versões associadas, além de possuir características próprias. Um objeto versionado pode apresentar particularidades que devem ser comuns a todas as suas versões, e cada uma só faz parte de um único objeto versionado. A figura 2.2 apresenta a entidade *automóvel* que possui o objeto versionado *Fiat-uno* e suas versões.

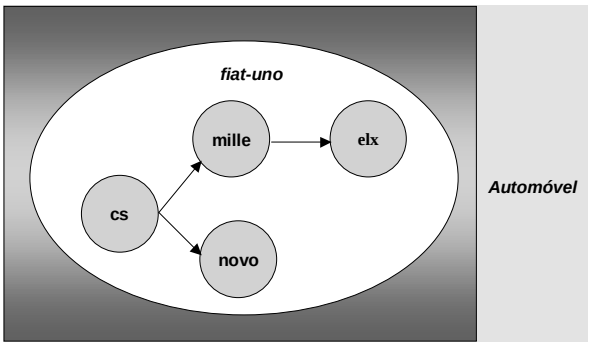


FIGURA 2.2 - Objeto versionado e suas versões

Cada objeto versionado possui uma versão que é considerada a corrente. A versão corrente é mantida automaticamente como a mais recentemente criada. O usuário pode especificar uma versão diferente para ser a corrente que, neste caso, ficará fixa, não mudando com o surgimento de novas versões. A versão corrente é utilizada sempre que o usuário solicitar uma operação sobre o objeto versionado, sem especificar uma determinada versão.

O versionamento pode ocorrer em diferentes níveis de abstração, onde cada um representa uma granularidade de versionamento. As versões são tratadas de maneira uniforme, sendo consideradas objetos versionáveis, e podem ser classificadas como segue (GOLENDZINER, 1995; GALANTE, 1998):

- versões de esquemas – são derivadas em decorrência de modificações realizadas na estrutura do esquema, como por exemplo, a eliminação de uma classe que desencadeia a derivação de uma versão completa do esquema, representando a atualização aplicada;
- versões de classes – são geradas por alterações realizadas em classes. Por exemplo, a inclusão de uma propriedade em uma classe provoca a derivação de uma nova versão desta classe contendo a modificação efetuada;

- versões de métodos – são derivadas para registrar as modificações realizadas no código e definições dos métodos. Entretanto, a inclusão e a exclusão de métodos provocam a derivação de versões de classe, uma vez que a versão de classe é alterada;
- versões de objetos – são definidas para fins de adaptação das instâncias presentes no banco de dados para que permaneçam em conformidade com as definições especificadas nas diversas versões de classes e esquemas.

Nesta seção foram apresentados os principais conceitos relacionados a versões e ao Modelo de Versões. O conceito de configuração será apresentado no capítulo 5, na seção 5.1.6, como uma característica importante do TVM.

2.3 Considerações finais

Neste capítulo foram apresentados os principais conceitos de tempo, bem como os tipos de bancos de dados temporais existentes. Quanto ao conceito de versão foram apresentadas as formas de representação da sua evolução (derivativas ou alternativas), as características de versão e objeto versionado e, por fim, uma classificação quanto à granularidade do versionamento. O TVM, modelo de dados utilizado na implementação do protótipo, utiliza os conceitos de versão e objeto versionado e une a eles características temporais.

No capítulo 3 são apresentadas as características necessárias para que ocorra a evolução de esquemas por meio de versionamento em um ambiente temporal.

3 Versionamento de esquemas

Desde a sua criação, um esquema conceitual está em constante alteração. A manutenção coerente dos esquemas alterados e dos dados referentes a estas versões dos esquemas é uma tarefa complexa. Esta dificuldade é aumentada quando se trata de bancos de dados temporais. Em Roddick et al. (2000) foram identificados seis tipos de mudanças que resultam na necessidade de evolução de um esquema:

- mudança no universo do discurso;
- mudança na interpretação dos fatos sobre o universo do discurso e na maneira pela qual a tarefa é realizada no sistema;
- mudanças na forma de atualizar informações para aumentar a eficiência;
- mudanças causadas por uma operação de sistema;
- correção de erros de projeto.

A alteração de um esquema pode ser classificada em três tipos diferentes: modificação, evolução, e versionamento do esquema. A seguir é apresentada a definição de cada um dos tipos citados (JENSEN ET AL, 1998; MOREIRA, 1999):

- modificação do esquema – ocorre quando o sistema de banco de dados permite a alteração da definição de um esquema com uma base de dados populada;
- evolução do esquema – ocorre quando o sistema de banco de dados permite a modificação do esquema, e esta mudança ocorre sem perdas de informações existentes;
- versionamento de esquema – ocorre quando o sistema de banco de dados permite a visualização de todos os dados, tanto retrospectiva como prospectivamente, por meio de interfaces para versões definidas pelo usuário.

O versionamento é a maneira mais completa e com menos riscos na realização de alterações no esquema. Para que o versionamento de esquemas possa ocorrer é necessário que os seguintes requisitos sejam atendidos (RODDICK, 1994; GRANDI, MANDREOLI, 1999; MANDREOLI, 2001):

- o processo de modificação do esquema e os detalhes sobre a manutenção de versões do esquema devem ser completamente gerenciados pelo sistema e transparentes aos usuários. A intervenção do administrador do banco de dados deve ser mínima;
- a modificação do esquema deve ser a mais simétrica possível, de tal forma que não somente os dados existentes possam ser visualizados por meio da nova definição do esquema, mas que também os dados armazenados após a modificação possam ser visualizados por meio de versões anteriores. A mudança deve ser o mais reversível possível, para que modificações errôneas possam ser removidas. Isto implica que as funções de modificação de dados devem operar sem perda de informação;
- mudanças em uma escala maior do que operações básicas devem ser compostas por operações elementares.

Em um contexto mais atual, versionamento de esquemas tem também sido utilizado por aplicações orientadas a objeto típicas como, por exemplo, multimídia e sistemas de informação geográfica (SIGs). Estas aplicações necessitam de

características temporais a fim de gerenciar e recuperar a evolução dos esquemas e dos objetos.

3.1 Características gerais

O versionamento de esquemas pode assumir diferentes características de acordo com a forma de versionar, com o momento da propagação das alterações, ou quanto à forma de armazenamento dos dados da extensão. Nas seções seguintes, cada uma destas características será abordada.

3.1.1 Meta-esquema

O principal objetivo de um meta-esquema é a manutenção de informações a respeito das modificações realizadas nos esquemas e dados, armazenando assim, todo o histórico da evolução. Toda derivação de versões resulta no armazenamento de informações no meta-esquema, e cada versão de esquema possui uma instância no meta-esquema. Quando uma nova versão de esquema é derivada, é feita também sua representação no meta-esquema.

O gerenciamento do versionamento de esquemas pode ser dividido em três níveis: meta-esquema, intenção e extensão. No nível do meta-esquema é realizado o controle da evolução de todas as versões de esquema, de seus tempos associados e das referências ao(s) lugar(es) onde estas estão armazenadas. No nível da intenção ou dos esquemas estão armazenadas, para cada versão de esquema, as informações que o compõem, como suas classes, atributos, relacionamentos, métodos e operações. O terceiro nível (extensão) contém as instâncias armazenadas de acordo com cada versão de esquema.

Para exemplificar a definição de um meta-esquema será apresentado o meta-esquema proposto em Roma et al. (2001) e Galante et al. (2002), onde cada classe do meta-esquema gerencia um determinado tipo de evolução (figura 3.1). A camada meta-esquema é definida por meio de classes que mantêm informações sobre a evolução dos esquemas, classes, atributos e relacionamentos, servindo para o processo de versionamento de esquemas.

A classe `Esquema` mantém informações sobre as versões de esquemas, armazenando o nome do esquema, o número do repositório (caso a solução de armazenamento seja a de múltiplos repositórios, seção 3.1.4), o formato temporal suportado (tempo de transação, validade ou bitemporal) e o tempo de vida da versão de esquema.

A classe `Classe` mantém informações sobre as classes de cada versão de esquema, armazenando o nome da classe e os seus respectivos rótulos temporais.

A classe `Atributo` mantém informações sobre os atributos das classes em cada versão de esquema, armazenando o nome do atributo, o tipo e seus rótulos temporais.

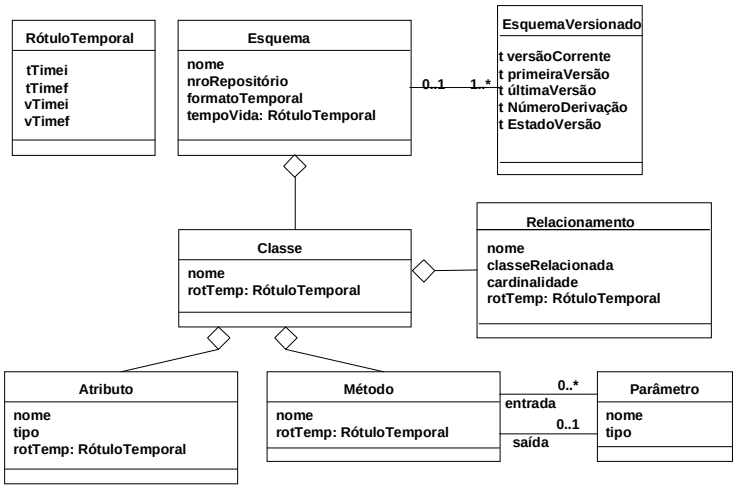


FIGURA 3.1 - Meta-esquema

A classe `Relacionamento` mantém informações sobre os relacionamentos entre as classes de cada versão de esquema, armazenando o nome do relacionamento, a classe relacionada, a cardinalidade do relacionamento e seus rótulos temporais.

As classes `Método` e `Parâmetro` armazenam informações quanto ao comportamento das classes, armazenando o nome do método e seus rótulos temporais. Mediante um relacionamento com a classe `Parâmetro` é possível verificar o nome e o tipo do parâmetro associado a um método.

A classe `EsquemaVersionado` realiza o armazenamento das informações históricas de todas as versões de esquema, armazenando os históricos da versão corrente, da primeira versão, da última versão, o número de derivações de determinada versão, e os estados das versões.

3.1.2 Modo do versionamento

A forma de versionamento pode ser parcial ou total. O versionamento parcial permite alterações somente a partir da versão corrente do esquema, que é a versão que está sendo usada para definir os dados da extensão. Normalmente, existe uma única versão corrente que geralmente corresponde à última derivada, porém pode existir mais de uma versão corrente. Neste caso, o usuário é quem as define.

No versionamento total são permitidas alterações sobre qualquer versão de esquema independente de ser a versão corrente. Devido a essas características, é possível associar a forma de representação do versionamento parcial como lista, e do total como grafo acíclico dirigido.

3.1.3 Alternativas de propagação das instâncias

A propagação das alterações do esquema sobre a base de dados pode ser classificada como *off-line* ou *on-line* (CAMOLESI, TRAINA, 1996).

Na estratégia *off-line*, todos os processos em andamento são interrompidos enquanto a base está sendo reorganizada. De acordo com esta característica, a estratégia *off-line* deve ser executada em um período de baixa requisição de dados porque, durante a propagação, os usuários terão seus pedidos de acesso negados.

Na estratégia *on-line*, o SGBD é mantido em operação enquanto realiza a propagação das alterações do esquema. Esta alternativa é recomendada para sistemas críticos ou que envolvam um grande volume de dados. A reorganização e a utilização são processos concorrentes e devem ser controlados de forma que o usuário possa ter

suas requisições de acesso a dados atendidas enquanto uma porção do banco de dados é atualizada.

3.1.4 Alternativas de armazenamento

O armazenamento dos dados da extensão pode ser feito de duas maneiras, de acordo com o número de repositórios de dados utilizado: um único ou múltiplos repositórios (CASTRO, GRANDI, SCALAS, 1995, 1997; CASTRO, 1997).

Repositório único

Nesta solução é utilizado somente um repositório onde todos os dados da extensão são armazenados de acordo com um esquema global, o qual inclui todos os atributos introduzidos pelas sucessivas mudanças no esquema, não podendo nenhum dado ser descartado do repositório.

Desta forma, o repositório nunca irá "encolher", mesmo na ocasião de uma modificação destrutiva, como por exemplo, a exclusão de um atributo, ou a restrição de um domínio. Estas modificações não podem ser executadas fisicamente, sendo apenas gravadas em um catálogo. Já as mudanças construtivas, como por exemplo, adição de um atributo, ou extensão de um domínio, fazem a conversão de todo o repositório para o novo formato.

As mudanças realizadas em um esquema onde a forma de armazenamento é por meio de um único repositório implicam em uma semântica para os valores nulos, ou seja, para os valores indefinidos num determinado ponto no tempo. Em Roddick (1994) foi realizado um estudo onde os atributos eram classificados quanto a serem aplicáveis ou não, conhecidos ou desconhecidos, ou ainda, definidos ou indefinidos nas diferentes versões de esquema. Resultam dessas combinações sete tipos diferentes de classificação quanto aos valores nulos, como ilustra a tabela 3.1.

TABELA 3.1 - Interpretação dos valores nulos

	Atributo é definido		Atributo não é definido	
	Valor é conhecido	Valor é desconhecido	Valor é conhecido	Valor é desconhecido
Atributo é Aplicável	Valor	ω_1	ω_4	ω_5
Atributo não é Aplicável		ω_2		ω_6
Aplicabilidade é Desconhecida	ω_3		ω_7	

Nesta proposta, a questão considerada é atribuir corretamente a semântica dos valores nulos, isto é, como estes valores serão retornados ao usuário final de um banco de dados temporal que suporta evolução de esquemas. Por exemplo, uma pessoa não deixa de ter data de nascimento mesmo que esta informação não seja mais necessária à aplicação, neste caso o valor "ω_1" deve ser utilizado. Outro exemplo é quando existe a evolução de uma determinada versão de esquema onde um atributo é definido para uma nova versão, mas não existe nas anteriores; nesta situação seria utilizado o valor "ω_5" nas versões anteriores pois o valor é desconhecido e o atributo é aplicável.

Múltiplos repositórios

Esta solução requer a criação de tantos repositórios quantas forem as versões do esquema. Neste caso, cada repositório é formado de acordo com a versão do esquema correspondente. Quando um novo repositório é criado, os dados do repositório antigo são copiados e adaptados para refletir a nova versão de esquema.

A solução de múltiplos repositórios implica na proliferação do número de repositórios de dados (tantos quanto o número de versões de esquemas) e em grande quantidade de dados duplicados.

A escolha da solução de múltiplos repositórios permite um certo grau de liberdade, pois a cada versão de esquema estão associados os dados correspondentes. Na solução de repositório único existe um crescimento no formato dos dados e um gerenciamento complexo dos valores nulos, mas são evitadas duplicações em grande escala.

3.1.5 Sincronismo entre esquemas e dados

Além do versionamento dos esquemas (intenção) existe o versionamento dos dados (extensão), que também são dependentes do tipo de rótulo temporal a eles associados. Estas evoluções são armazenadas em bancos de dados distintos, isto é, a evolução dos dados é conceitualmente independente da evolução dos esquemas. O esquema conceitual deve permanecer consistente com a base de dados. Para garantir este requisito, deve-se definir o sincronismo entre o esquema e os respectivos dados armazenados. Existem duas estratégias que permitem este gerenciamento: síncrono e assíncrono.

Gerenciamento síncrono

Neste tipo de gerenciamento, a pertinência temporal de uma versão do esquema deve incluir a pertinência temporal dos dados correspondentes, tornando-os dependentes. Os dados são armazenados, recuperados e atualizados sempre com a versão do esquema que tenha a mesma pertinência temporal. Isto é, de acordo com a versão do esquema são recuperados os dados correspondentes.

Gerenciamento assíncrono

Neste tipo de gerenciamento, a pertinência temporal de uma versão do esquema e a pertinência temporal dos dados correspondentes são completamente independentes. Os dados podem ser recuperados e atualizados mediante qualquer versão de esquema cuja validade é independente da validade dos dados, mesmo com dimensões temporais comuns.

3.2 Tipos de versionamento

Dependendo do tipo de banco de dados temporal utilizado, existirão diferentes tipos de versionamento de esquemas, como por exemplo, o versionamento de tempo de transação, de tempo de validade e bitemporal. A cada modificação do esquema são acrescentadas informações temporais referentes à alteração. Estas informações adicionais irão variar conforme o suporte temporal desejado (MOREIRA, 1999; ANGONESE, 2000).

De acordo com o tipo de banco de dados temporal é necessária a aplicação de restrições no que diz respeito às modificações do esquema, em especial nos bancos de dados de tempo de validade e bitemporal. Estas restrições referem-se à natureza do tempo de validade. Considerando as situações em que o esquema evolui, não é possível

nem significativa uma alteração retrotativa nos dados da base. A modificação do tempo de validade no que diz respeito aos esquemas deve estar relacionada ao momento atual ou futuro, fazendo com que as partes alteradas do esquema se tornem válidas somente quando a validade da nova versão do esquema for atingida (EDELWEISS, CASTILHO, OLIVEIRA, 1995).

3.2.1 Versionamento de esquemas de tempo de transação

Neste tipo de versionamento, as versões de esquema são identificadas pelo tempo de transação, isto é, o momento em que são criadas. A última versão definida possui o tempo de transação final igual a `nulo`, ou seja, está representando o momento presente. Não são permitidas versões retro ou proativas.

A identificação dos dados correspondentes a cada versão do esquema pode ser realizada pela inclusão do tempo de transação do esquema em cada instância. A figura 3.2 ilustra o armazenamento em um único repositório realizado num banco de dados de tempo de transação.

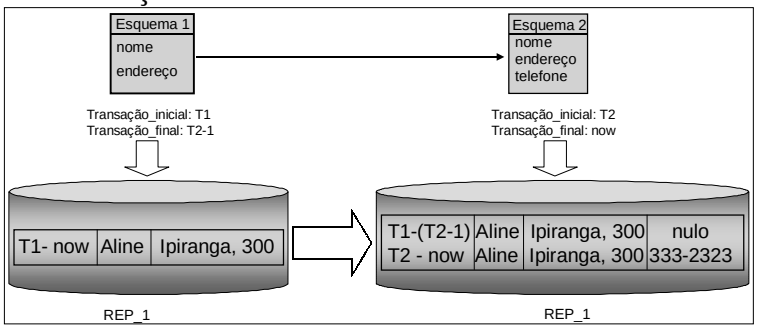


FIGURA 3.2 - Exemplo de armazenamento em um repositório único (esquema completado)

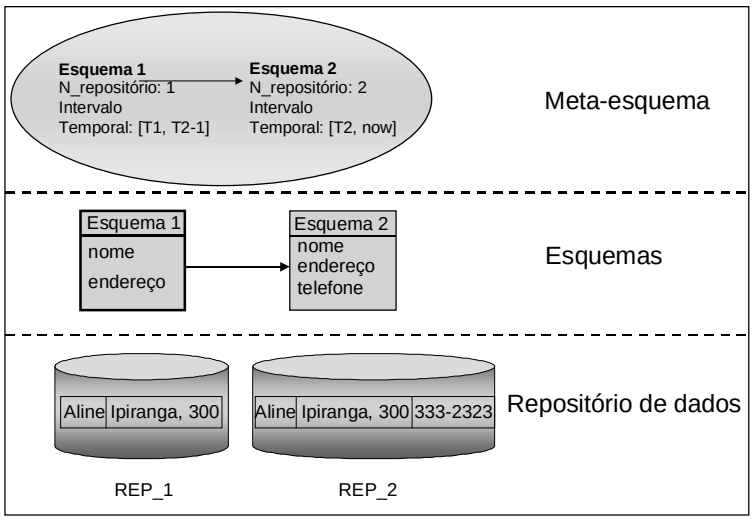


FIGURA 3.3 - Exemplo de armazenamento em múltiplos repositórios

Em contrapartida, se a solução escolhida para o armazenamento for a de múltiplos repositórios, não é necessário armazenar o tempo de transação em cada instância. Neste caso, para cada esquema é registrado no meta-esquema o número do repositório onde os dados correspondentes estão armazenados. Na figura 3.3 é ilustrada esta situação.

3.2.2 Versionamento de esquemas de tempo de validade

Quando o esquema conceitual é representado por um banco de dados de tempo de validade, cada modificação no esquema é rotulada com o tempo de validade correspondente, não sendo o tempo de transação armazenado. A nova versão do esquema torna-se válida somente quando o tempo de validade é alcançado, independente do tempo de transação. Se várias alterações possuem a mesma validade, elas são executadas em conjunto, e a consistência do esquema é analisada uma única vez.

No entanto, existe uma forte restrição a essas alterações de esquema, pois o tempo de validade de uma alteração deve ser maior do que tempo atual (EDELWEISS, CASTILHO, OLIVEIRA, 1995). Alterações no esquema passado não fazem sentido, pois um esquema modela a realidade atual. O SGBDT deve prover mecanismos para que uma alteração de esquema que não represente a realidade correta possa ser retornada à representação passada.

3.2.3 Versionamento de esquemas bitemporal

No versionamento de esquemas bitemporal, a cada alteração efetuada no esquema são associados os tempos de transação e de validade. O tempo de transação informa quando foi efetuada a modificação e o tempo de validade define o instante a partir do qual a alteração do esquema se torna válida. Este tipo de versionamento permite a manutenção de todas as versões de tempo de validade inseridas nas sucessivas mudanças no esquema. Considerando o armazenamento da extensão em um repositório único, os dados são armazenados de acordo com seu esquema completado.

A utilidade deste tipo de representação temporal nas versões de esquemas é a possibilidade de conhecer o momento em que o esquema foi modificado independentemente do tempo em que suas modificações tornaram-se efetivas, sendo possível a recuperação tanto do histórico das transações quanto das validades (EDELWEISS, CASTILHO, OLIVEIRA, 1995).

3.3 Formas de derivação através do versionamento de esquemas

A evolução de esquemas com uso de versões sobre um banco de dados temporal pode acontecer de quatro formas diferentes. As diferenças consistem na escolha da versão que gera a derivação e na definição de qual versão será considerada a versão corrente.

3.3.1 Alternativa de derivação 1

Nesta alternativa, a derivação sempre ocorre a partir da última versão que também é a versão corrente. No momento da derivação, a nova versão de esquema passa a ser a versão corrente. Uma única versão poderá ser a corrente a cada instante, pois esta alternativa de versionamento se caracteriza por utilizar uma única linha de tempo (versionamento de esquemas com tempo linear). A representação deste tipo de versionamento está ilustrada na figura 3.4, sendo a versão corrente (e_3) identificada por um contorno mais forte.

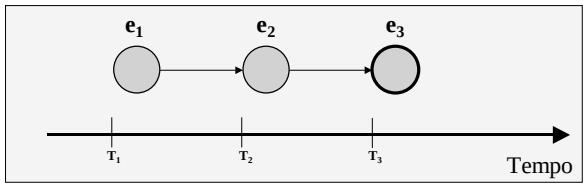


FIGURA 3.4 - Forma de derivação da alternativa 1

Para exemplificar a utilização desta alternativa de derivação podemos imaginar que uma versão de esquema seja a solução proposta para um determinado problema, e que as versões derivadas desta são melhoramentos realizados a fim de resolver plenamente o problema proposto, sendo as alterações realizadas a partir da última versão de esquema.

3.3.2 Alternativa de derivação 2

Nesta alternativa não necessariamente a última versão é a que pode sofrer derivação; qualquer versão pode servir de base para derivação, mas somente uma é considerada a versão corrente, que neste caso é sempre a última versão derivada. Semelhante à alternativa 1, este versionamento também é de tempo linear. A figura 3.5 apresenta a forma de derivação desta alternativa.

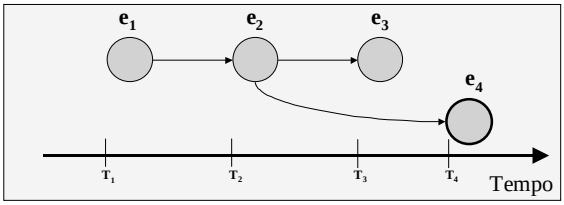


FIGURA 3.5 - Forma de derivação da alternativa 2

De forma semelhante a primeira alternativa, também podemos considerar cada versão de esquema como a solução proposta para um determinado problema, e que as versões derivadas são alternativas propostas para a solução deste. Neste caso é possível retomar uma solução anterior e evoluir a partir desta.

3.3.3 Alternativa de derivação 3

Também nesta alternativa não necessariamente a última versão é a que pode sofrer derivação; qualquer versão pode servir de base para derivação. Diferenciando da alternativa anterior, qualquer versão pode ser considerada a corrente, desde que seja somente uma por vez. Semelhante às alternativas 1 e 2, este versionamento também é de tempo linear. A figura 3.6 apresenta a forma de derivação desta alternativa, onde a versão corrente é a e_2.

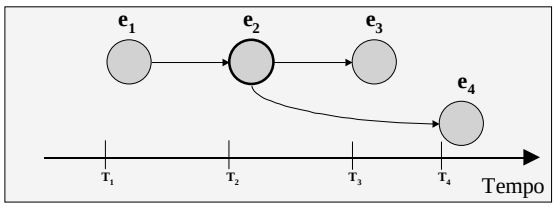


FIGURA 3.6 - Forma de derivação da alternativa 3

De forma semelhante a segunda alternativa é possível retomar uma solução anterior e evoluir a partir desta, e o usuário pode definir uma versão antiga como a corrente.

3.3.4 Alternativa de derivação 4

De forma semelhante às alternativas 2 e 3, nesta alternativa a derivação também pode ocorrer a partir de qualquer versão de esquema. E também, como na alternativa 3 qualquer versão de esquema pode ser considerada a versão corrente. Mas agora, mais de uma versão pode estar ativa, ou seja, podem existir diversas versões correntes. Cada uma destas versões correntes corresponderia a uma aplicação diferente, com seus dados associados, todas funcionando em paralelo. Devido a esta característica o versionamento é classificado como sendo de tempo ramificado. Na figura 3.7 as versões e_3 e e_4 seriam as versões correntes.

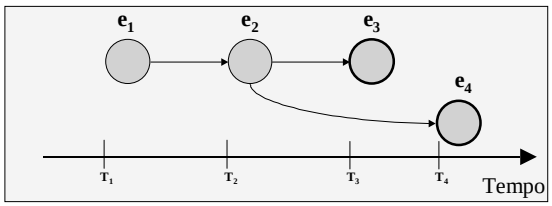


FIGURA 3.7 - Forma de derivação da alternativa 4

Este é um caso bem particular, podemos exemplificar a sua utilização como soluções distintas para um determinado problema. Por exemplo, duas cidades são separadas por um rio, e são propostas duas soluções alternativas: uma ponte ou um túnel. O estudo de cada solução se dará em separado, caracterizando a existência de mais de uma versão corrente ao mesmo tempo.

3.4 Operações de modificação de esquemas

Em Moreira (1999) foi realizado um estudo das operações de modificação de esquemas nos modelos relacional, entidade-relacionamento e orientado a objetos. Na implementação realizada na presente dissertação é simulado o versionamento de esquemas de um modelo orientado a objetos sobre um banco de dados relacional. Devido a esta característica serão apresentadas aqui as operações de modificação de esquemas nos modelos relacional e orientado a objetos.

3.4.1 Modelo relacional

O esquema conceitual do modelo relacional contém as definições de tabelas, as quais compreendem as definições de atributos pertencentes a um domínio simples. Diversas modificações podem ser realizadas durante a evolução de um esquema relacional. As principais alterações são em atributos, em relações, ou ainda, em atributos e relações simultaneamente. A tabela 3.2 apresenta as principais operações de modificação que podem ocorrer no modelo relacional.

TABELA 3.2 - Operações de modificação de esquema no Modelo Relacional
(continua)

Categoria	Primitiva de alteração
Mudanças em atributos	
	Expansão do domínio
	Restrição do domínio
	Mudança no domínio
Mudanças em relações	
	Acrescentar uma relação
	Desativar uma relação

TABELA 3.2 - Operações de modificação de esquema no Modelo Relacional (continuação)

Categoria	Primitiva de alteração
Mudanças em atributos/relações	
	Acrescentar um atributo a uma relação
	Renomear um atributo
	Desativar um atributo não-chave[1]
	Promover um atributo
	Rebaixar um atributo
	Dividir uma relação
	Particionar uma relação
	Juntar duas relações

3.4.2 Modelo orientado a objetos

Um esquema orientado a objetos contém uma coleção de classes organizadas em forma de hierarquia. Objetos são criados como instâncias das classes, encapsulando os dados e sua interpretação (RA, RUNDENSTEINER, 1995; MANDREOLI, 2001). A tabela 3.3 apresenta as possíveis operações de modificação de esquemas num modelo orientado a objetos.

TABELA 3.3 - Operações de modificação de esquema no Modelo Orientado a Objetos

Categoria	Primitiva de alteração
Mudanças em classes - Mudanças nos atributos de uma classe	
	Incluir um atributo a uma classe
	Excluir um atributo de uma classe
	Renomear um atributo
	Mudar o domínio de um atributo
	Mudar o controle de acesso
Mudanças nos métodos de uma classe	
	Incluir um método
	Excluir um método
	Renomear um método
	Mudar o tipo de argumentos de um método
	Mudar o tipo de retorno de um método
	Mudar o controle de acesso
Mudanças nos relacionamentos de hierarquia	
	Incluir uma superclasse
	Excluir uma superclasse
Mudanças numa coleção de classes	
	Incluir uma classe
	Excluir uma classe
	Renomear uma classe

Não necessariamente todas as operações geram novas versões de esquemas. Operações como mudança de nome de uma classe, atributo ou método podem acarretar apenas em uma correção. Para assegurar a correção das operações de modificação deve-se definir restrições de integridade. Em Edelweiss, Castilho e Oliveira (1995) foram relacionadas as invariantes necessárias à evolução de esquemas em um modelo orientado a objetos genérico:

- unicidade dos nomes – os nomes das classes, de suas propriedades e de métodos devem ser únicos. Se a classe é definida como subclasse, herda

[1] Quando for necessário desativar um atributo chave, ele primeiro deve ser rebaixado.

as propriedades de sua superclasse e as novas propriedades definidas devem ter nomes únicos;

- todas as propriedades devem ter um domínio definido;
- se uma classe é definida como subclasse, a superclasse correspondente precisa ser definida;
- se uma classe é definida como uma agregação de um conjunto de classes, os componentes das classes precisam ser definidos.

Operações complexas para a modificação de esquemas

Operações complexas se caracterizam por envolver uma ou mais operações simples na sua execução. Exemplos de operações complexas são: união de classes em uma nova classe, decomposição de uma classe em novas classes, criação de uma nova classe como superclasse generalizando outras existentes. A tabela 3.4 apresenta uma relação das operações complexas (ROMA, 2000).

TABELA 3.4 - Operações complexas de evolução de esquemas

Primitiva de alteração	Significado
Mudanças em classes	
Transferir_atributo	Transfere um atributo de uma versão de classe para outra
Mudanças na estrutura do esquema	
Unir_classes	Une classes em uma superclasse, criando uma nova versão
Unir_classes_intersecção_propriedades	Faz a intersecção de propriedades de uma classe em uma superclasse, criando uma nova versão
Unir_classes_diferença_propriedades	Faz a diferença de propriedades de uma classe em uma superclasse, criando uma nova versão
Generalizar_classes	
Especializar_classes	
Decompor_classes	
Excluir_composição_incluindo_propriedades	
Decompor_classe	
Excluir_composição_incluindo_propriedades	
Decompor_classe_criando_agregação	
Decompor_classes_criando_agregações	
Agregar_classes	
Excluir_classe_propagando_propriedades	
Excluir_classe_recompondo_hierarquia	
Excluir_classe_propagando_recompondo	
Excluir_classe_em_cascata	
Operações de alteração no comportamento das classes	
Transferir_método	Transfere um método de uma versão de classe para outra.

3.5 Considerações Finais

Neste capítulo foram apresentadas as características de um ambiente temporal com suporte ao versionamento de esquemas. Dentre elas cabe salientar a especificação de um meta-esquema responsável pelo gerenciamento da evolução, as formas de versionamento, as alternativas de propagação das instâncias e de armazenamento, o tipo de gerenciamento entre os esquemas e sua base, as peculiaridades do versionamento quando diferentes dimensões temporais são consideradas, e as alternativas de derivação quando se considera o versionamento de esquemas. Além das características principais para a especificação de um ambiente temporal com suporte ao versionamento de esquemas, também foram apresentadas as operações que geram modificações em um esquema.

As operações de modificação de esquemas foram classificadas de acordo com o modelo de dados utilizado, relacional ou orientado a objetos. Estes dois modelos foram apresentados porque o TVM é um modelo orientado a objetos e a implementação do protótipo foi realizada sobre a parte relacional do banco de dados comercial DB2.

No próximo capítulo é apresentada uma revisão de trabalhos propostos para o gerenciamento da evolução de esquemas em bancos de dados temporais. E, no capítulo 6, é apresentada a análise sobre o versionamento de esquemas considerando o TVM como modelo de dados utilizado na implementação do protótipo.

4 Trabalhos relacionados

Este capítulo apresenta os trabalhos realizados na área de evolução de esquemas com o uso da técnica de versionamento, tendo como objetivo a apresentação das características de cada modelo proposto.

4.1 Proposta de Liu

Em Liu (1997) é apresentado um *framework* genérico para evolução de esquemas e adaptação de instâncias baseado em versões de esquema chamado DB-EVOLVE.

A evolução ocorre mediante a derivação de versões de esquema, onde a versão derivada herda o "instantâneo" da original no instante da derivação. Isto significa que todas as atualizações posteriores que ocorrerem sobre a versão de origem como uma inserção, uma exclusão ou uma alteração são transparentes para a versão de esquema derivada.

Para o controle da evolução é associado, a cada versão de esquema, um estado, que pode ser `released` ou `transient`. Uma nova versão de esquema recebe inicialmente o estado `transient`, e deve ser derivada de uma versão esquema `released`. Não existem limitações quanto ao número de versões de esquema que podem ser derivadas. Uma versão de esquema só pode ser excluída quando não possuir versões filhas, as quais podem ser tanto `transient` como `released`. No momento da exclusão de uma versão de esquema, além da sua exclusão também é eliminado o seu escopo de acesso às instâncias, sendo que elas permanecem armazenadas na base.

O DB-EVOLVE tem como base a orientação a objetos, permitindo assim o relacionamento de herança. Observando esta característica, as alterações em uma única classe podem atingir todas as subclasses da classe alterada.

O DB-EVOLVE suporta as seguintes primitivas para a evolução de esquemas:

- inclusão de uma propriedade;
- exclusão de uma propriedade;
- troca de nome de uma propriedade;
- alteração do domínio do tipo de uma propriedade ou a assinatura de um método;
- inclusão de uma classe à lista de superclasses;
- exclusão de uma classe da lista de superclasses;
- inclusão de uma classe;
- exclusão de uma classe;
- troca de nome de uma classe.

Toda vez que uma primitiva de evolução é requisitada esta gera a verificação das invariantes do *framework*. As invariantes são cinco e têm a finalidade de assegurar a correção e a consistência da evolução segundo o DB-EVOLVE:

1. invariante de unicidade de nome – cada classe deve ter um nome único. Cada instância variável e método definido ou herdado por uma classe deve ter um nome único;
2. invariante de subclasse e superclasse – o relacionamento entre subclasse-superclasse é do tipo `IS-A` (herança por refinamento) com a classe definida tendo a classe `OBJECT` como raiz;
3. invariante dos tipos das variáveis – o tipo de cada variável de instância deve ter uma classe correspondente na classe de definição do sistema;

4. invariante de herança – uma classe herda todas as propriedades (variáveis de instância e métodos) de suas superclasses, a menos que redefina uma propriedade com o mesmo nome. Quando mais de uma superclasse determina a mesma propriedade nome, a classe deve herdar somente o que foi definido pela superclasse que aparece primeiramente na lista de classes da superclasse;
5. invariante de compatibilidade de tipos – quando uma classe C_i define uma variável de instância com o mesmo nome que, em caso contrário, poderia ser herdada de uma de suas superclasses C_j, o tipo das variáveis de instância de C_i deve ser uma subclasse do tipo das variáveis de instância C_j.

A técnica de propagação de instâncias utilizada possibilita a herança automática do escopo de acesso da versão origem dentro da versão derivada. Isto ajuda a evitar cópias desnecessárias daqueles objetos da versão origem tornando-os visíveis para a versão derivada.

4.2 Proposta de Rodríguez

O modelo TVOO (*Temporal Versioned Object-Oriented Data Schema Model*) proposto por Rodríguez, Ogata e Yano (1999; 2000), possibilita a evolução de versões temporais de esquemas. O elemento versionado é o instantâneo de um objeto em um certo tempo, e muitas versões podem coexistir representando diferentes estados paralelos de um mesmo objeto.

O versionamento é implementado por meio de uma estrutura de árvores onde cada versão corresponde a um elemento (nodo) da árvore, com comportamentos independentes, bem como identificadores de versões e rótulos temporais associados.

No TVOO, o esquema é dependente do tempo e da história das mudanças que ocorrem em seus elementos. Estas informações são mantidas dentro de hierarquias de versões. Uma nova especificação de esquema não deve definir uma nova base de dados. As características anteriores de esquema são consideradas como especificações de alternativas de projeto e os dados existentes, conseqüentemente, podem ser acessados de forma consistente usando qualquer um dos esquemas definidos.

Apenas uma base de dados pode ser acessada por qualquer versão do esquema (repositório único), e o esquema corrente é o único esquema válido. As mudanças não são consideradas corretivas, e os esquemas não ficam obsoletos após a mudança. O TVOO apresenta gerenciamento assíncrono, onde diferentes esquemas podem coexistir com a finalidade de manipulação da base de dados.

A granularidade utilizada para o versionamento de esquema é o esquema, e a granularidade temporal utilizada é o minuto. Neste contexto, cada versão corresponde a um elemento da árvore e seu comportamento é independente dos outros. Cada versão possui somente um identificador, um tempo de criação e possivelmente um tempo de exclusão; durante a sua existência, as versões são denominadas “vivas”. O tipo de exclusão permitida é a lógica, portanto uma versão excluída pode retornar à árvore de derivação desde que não haja sobreposição dos períodos de vida desta versão.

O tempo utilizado no TVOO corresponde ao de transação, assumido como discreto, isto é, o tempo é considerado como uma sucessão de inteiros não negativos na sua ordem usual.

No TVOO todos os objetos são temporais (ou históricos), isto é, seus valores podem ser alterados e os diferentes valores são armazenados no banco de dados de acordo com o instante da alteração. O modelo gerencia o versionamento de esquemas no nível das classes. Isto torna possível a derivação de versões de esquema como

resultado de qualquer alteração na classe. O TVOO permite hierarquias de classes, ou seja, as classes podem ser definidas em termos de outras. No entanto, para o gerenciamento do versionamento de classes no TVOO, elas são agrupadas num tipo de hierarquia que é a combinação das hierarquias de herança e de versões de classe.

4.3 Proposta de Wei e Elmasri

Neste artigo (WEI, ELMASRI, 2000) os autores apresentam três técnicas de versionamento em bancos de dados bitemporais num ambiente relacional, STV (*Single Table Version*), MTV (*Multiple Table Version*) e PMTV (*Partial Multiple Table Version*).

Na primeira técnica (STV), cada tabela possui uma única versão durante a vida do banco de dados. Para a manutenção do histórico de todas as mudanças é proposto um esquema completado, o qual segue a idéia de tabela completada.

Um esquema completado consiste de tabelas definidas sobre a união dos atributos que foram definidos para ele. No caso de exclusão, o atributo eliminado continua armazenado no banco de dados somente para consultas, pois em bancos de dados temporais os dados nunca são excluídos. Além do mais, o tamanho do registro e o conseqüente tamanho da tabela para esta abordagem só poderão crescer e nunca diminuir.

Três problemas são verificados neste enfoque: aumento de espaço e de tempo de pesquisa, e a possível implementação do banco de dados. O banco de dados sempre é convertido para comportar a mudança de esquema. Cada vez que um atributo é incluído ou excluído de uma tabela, todas as versões correntes das entidades precisam ser verificadas para análise de seus intervalos de validade quanto à sua sobreposição na mudança do esquema. Para bancos de dados temporais, os quais usualmente contêm um grande volume de dados, o tempo de pesquisa terá uma grande sobrecarga. Para o problema da viabilização de um banco de dados, quando um novo atributo é incluído ou o tipo de domínio de um atributo é generalizado, parte do banco de dados (ou mesmo todo) não estará disponível por um período de tempo em paralelo ao processo de conversão do banco de dados, o que requer acréscimo e reorganização do espaço de armazenamento.

Na segunda técnica, *Multiple Table Version* (MTV), cada vez que uma tabela do esquema é alterada é criada uma nova versão da tabela. As versões de entidades correntes são a tabela fonte cujo intervalo de tempo de validade se sobrepõe ao intervalo da mudança de esquema, devendo ser copiados para a nova tabela, adaptados à nova versão do esquema.

Três problemas são encontrados nesta abordagem: duplicação de dados, consultas multiesquema e criação de versões obrigatórias.

A terceira técnica, *Partial Multiple Table Version* (PMTV), usa o conceito de normalização temporal. Para a mudança do esquema de uma inclusão de atributo, desde que o novo atributo incluído não possa ser sincronizado com os existentes na tabela, a abordagem PMTV cria uma tabela bitemporal somente com os novos atributos, mais o atributo chave (`ID`) da tabela que está sendo modificada. A tabela completa pode ser depois reconstruída pela aplicação da operação **`join`** de entidade.

A abordagem PMTV resolve os problemas que as abordagens STV e MTV apresentaram, porém, segundo os próprios autores, ainda existe a necessidade de testes para a comprovação de sua correção e performance em relação às demais.

4.4 Proposta de Mandreoli

Na tese de Federica Mandreoli é proposto um modelo de dados orientado a objetos para evolução de esquemas chamado OODM$|_{SV}^{+}$ (MANDREOLI, 2001; GRANDI, MANDREOLI, 2002).

OODM$|_{SV}^{+}$ modela um cenário onde diferentes versões de esquema consolidadas são armazenadas junto com o histórico de todas as suas mudanças, com a possibilidade de derivar uma nova versão consolidada de outra já existente. A granularidade do versionamento é a versão de esquema.

A evolução do esquema é representada por meio de um grafo acíclico dirigido (GAD), onde os nodos representam versões consolidadas e os arcos representam os relacionamentos de herança. Todos os elementos do grafo são rotulados com tempo de transação a fim de manter o histórico das alterações do esquema.

O conjunto de primitivas de mudanças no esquema é composto de dois subconjuntos, de mudanças nos conteúdos de um nodo e mudanças nos elementos (nodos e arestas) do grafo. Primitivas de mudanças são classificadas nas seguintes categorias: modificações nas classes, nos métodos, nos relacionamentos hierárquicos e nas coleções de classes, conforme apresentado na tabela 4.1.

TABELA 4.1 - Primitivas de modificação de uma versão de esquema no OODM$|_{SV}^{+}$

Modificações nas classes	
`addAttribute`	acrescenta um atributo para um tipo
`deleteAttribute`	exclui um atributo de um tipo
`changeAttrName`	muda o nome de um atributo
`changeAttrType`	muda o tipo de um atributo
`changeClassType`	muda o tipo de classe
Modificações nos métodos	
`addMethod`	acrescenta um método a uma classe
`deleteMethod`	exclui um método de uma classe
`changeMethodName`	muda o nome do método
`changeMethodCode`	muda a implementação do método
Modificações nos relacionamentos hierárquicos	
`addSuperclass`	torna uma classe existente em uma superclasse
`deleteSuperclass`	exclui uma classe da superclasse
Modificações na coleção de classes	
`AddClass`	acrescenta uma nova classe isolada vazia
`DeleteClass`	exclui uma classe isolada
`changeClassName`	muda o nome de uma classe

Cada alteração no esquema gera uma nova versão. Na maioria dos casos, as mudanças precisam ser propagadas para as instâncias a fim de assegurar a consistência entre a nova versão de esquema e seus dados. Com este propósito, foram inseridas duas funções, a primeira chamada `Schema Version Update`, cujo comportamento especifica como cada modificação no esquema gera uma nova versão, e a segunda chamada `Instance Update`, cujo comportamento especifica como as instâncias devem ser modificadas para se tornarem consistentes com a versão do esquema resultante.

Para a manutenção da correção no processo de atualização do esquema foi definido um conjunto de invariantes responsáveis pela verificação dos domínios dos tipos, por evitar a ocorrência de ciclos, e pela verificação da unicidade dos nomes dentro das versões de esquema, nos seus atributos e operações.

Além das primitivas de alteração do esquema foram definidas primitivas de alteração dos grafos, que têm como finalidade a criação ou a exclusão de um novo nodo ou aresta, e a integração de uma ou mais versões de esquema (`mergeVersion`).

A nova versão de esquema pode ser criada por meio das especificações do usuário, o que é considerado como um novo nodo isolado, ou pode ser derivada de uma outra versão existente.

A exclusão implica na remoção do nodo do banco de dados corrente. Isto é realizado por intermédio do encerramento dos tempos de transação de todas as versões de esquema pertencentes àquele nodo e do próprio nodo no grafo de derivação.

4.5 Proposta de Galante et al.

A proposta de Galante et al. (2002) apresenta um ambiente para a evolução de esquemas sobre o TVM[2] tendo como base o estudo de Roma et al. (2001). Nessa proposta, a representação da evolução de esquemas é realizada por meio de estados, que são alterados por intermédio de operações sobre o esquema. Uma versão de esquema é criada inicialmente no estado `em trabalho`. Neste estado, a versão representa um esquema que está em fase de definição (todas as alterações causam correções). Não possui tempo associado, não pode ser instanciada ou referenciada, e pode ser removida fisicamente. A versão `em trabalho` pode ser promovida a `estável` explicitamente, ou é promovida automaticamente no momento em que servir de base para a definição de outra versão de esquema. A partir deste estado esta versão não pode mais ser modificada. Pode ser instanciada, referenciada e servir de base para a definição de outras versões. Pode também, ser promovida a consolidada. Nesse estado a versão não pode mais ser instanciada, mas pode ainda ser utilizada para gerar novas versões. Pode, ainda, voltar ao estado `estável`. Uma versão no estado `estável` ou `consolidada` pode ser desativada, passando ao estado `congelada` que corresponde a exclusão lógica. As versões de esquema no estado `congelada` podem ser utilizadas somente para consultas. A figura 4.1 ilustra o diagrama de estados de uma versão de esquema TVM.

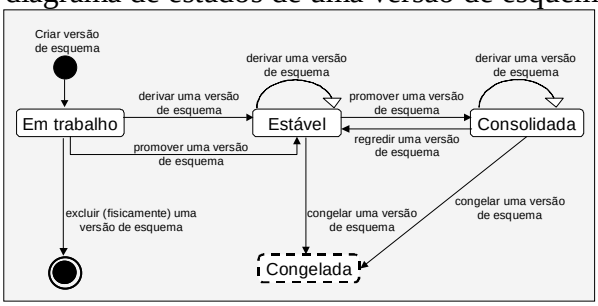


FIGURA 4.1 - Diagrama de estados de uma versão de esquema

Para que um esquema passe de um estado para outro são utilizadas operações de modificação. Estas operações têm como finalidade realizar ações como a criação de uma versão, uma derivação, o congelamento, ou a promoção de um estado para outro mais avançado, entre outras. A tabela 4.2 apresenta em quais estados de uma versão de esquema as operações podem ser aplicadas.

[2] Este modelo é apresentado com maiores detalhes no capítulo 5.

TABELA 4.2 - Operações sobre os estados das versões de esquema

Operações \ Estados	Em trabalho	Estável	Consolidada	Congelada
Instanciar		X		
Consultar	X	X	X	X
Alterar	X			
Derivar	X	X	X	
Excluir	X			
Congelar		X	X	

A seguir são descritas as operações sobre as versões de esquema:

- `criar` – permite a criação de uma nova versão de esquema;
- `derivar` – permite gerar uma nova versão de esquema derivada de uma ou mais versões existentes;
- `promover` – permite a promoção de uma versão de esquema de um estado para outro mais avançado;
- `regredir` – permite que uma versão consolidada retorne ao estado estável;
- `congelar` – permite a exclusão lógica de uma versão de esquema, sendo que esta versão continua disponível para consultas;
- `excluir` - permite a remoção física de uma versão de esquema.

A manutenção do histórico da evolução de esquema bem como a propagação das alterações nas instâncias da base é realizada mediante os seguintes critérios:

- primitivas de atualização – um grupo de operações de alteração do esquema. As primitivas são classificadas de acordo com o tipo de mudança realizada na versão de esquema e são classificadas em três categorias distintas: mudanças na estrutura do esquema, mudanças na estrutura das classes e mudanças no comportamento das classes;
- invariantes – são regras associadas às primitivas de atualização que asseguram a correção nas mudanças de esquema. Quando um erro é detectado, a primitiva de atualização é cancelada e o usuário é notificado;
- gerenciador da evolução – controla o gerenciamento da evolução de esquema, separando as mudanças de esquema da propagação dos dados;
- aplicações do usuário – uma base de dados armazena as aplicações do usuário e seus dados associados.

É utilizado o versionamento total como forma de derivação de versões de esquemas. As primitivas de atualização podem ser aplicadas a qualquer versão de esquema. Sempre que ocorre uma modificação no esquema é derivada uma nova versão a fim de manter o histórico das alterações. Tempo de transação e de validade são associados a cada versão de esquema caracterizando assim o versionamento bitemporal.

A consulta aos dados é sempre realizada de acordo com a versão de esquema correspondente. É proposta a solução de múltiplos repositórios como forma de armazenamento para os dados das versões de esquemas. Qualquer operação de modificação acarreta derivação de uma nova versão de esquema, o que leva à criação de um novo repositório de dados. Uma outra alternativa de armazenamento proposta

consiste no uso de um único repositório para o versionamento das classes. Esta abordagem permite evitar a proliferação de novas versões de esquema a cada mudança nas classes, a não ser que a modificação realizada seja significativa na estrutura do esquema.

A propagação das instâncias durante a evolução de esquemas sob o TVM é muito complexa. Por exemplo, se a primitiva de exclusão é aplicada em uma classe que possui subclasses, sua propagação às instâncias pode ser extremamente complexa porque esta mudança envolve alterações na hierarquia de herança. Neste caso, todas as conseqüências precisam ser analisadas para assegurar a consistência durante a propagação das alterações. Em Galante, Edelweiss e Santos (2002) foi acrescentado ao Modelo a especificação genérica do comportamento de funções de propagação e de conversão que serão utilizadas para a adaptação entre as mudanças no esquema e em seus dados associados.

O gerenciamento do mecanismo de evolução é dividido em três partes distintas: meta-esquema, intenção e extensão. O meta-esquema é definido por meio de quatro classes que mantêm informações sobre evolução dos esquemas, classes, atributos e relacionamentos, servindo para o processo de versionamento de esquemas. A intenção armazena as versões de esquema, cujas instâncias são obtidas nos dados da extensão. Dados da extensão podem ser organizados e gerenciados por meio da junção de um único e de múltiplos repositórios.

4.6 Considerações finais

O *framework* proposto por Liu (1997) suporta versões de esquema, mas não contempla o aspecto temporal nas suas características. Uma particularidade interessante é a solução para a adaptação de instâncias realizada por meio da herança do escopo de acesso do instantâneo da versão origem para a versão derivada.

Os demais modelos apresentados utilizam, além de versões, o tempo para a evolução de esquemas. A tabela 4.3 mostra um comparativo das características das propostas apresentadas.

TABELA 4.3 - Comparativo entre as propostas apresentadas

	A	*B*	*C*	*D*	*E*
Modelo de dados	*OO*	*OO*	*Relacional*	*OO*	*OO*
Gerenciamento	*Síncrono*	*Assíncrono*	-	-	*Síncrono*
Estratégia de armazenamento	*Repositório único*	*Repositório único*	-	-	*Múltiplos repositórios*
Adaptação das instâncias	*Herança do escopo de acesso*	-	*Conversão da base*	*Uso de funções*	*Uso de funções*
Representação da evolução	-	*Árvore*	-	*GAD*	*GAD*
Definição de invariantes	*Sim*	-	-	*Sim*	*Sim*
Estados das versões	*Released ou transient*	-	-	*Consolidada*	*Em trabalho, estável, consolidada e congelada*
Dimensão temporal suportada	-	*Tempo de transação*	*Tempo de transação e de validade*	*Tempo de transação*	*Tempo de transação e de validade*

A – Liu, 1997; B – Rodríguez, Ogata, Yano, 2000; C – Wei, Elmasri, 2000;
D – Mandreoli, 2001; E – Galante et al., 2002; "-" – não ou não consta.

O protótipo implementado nesta dissertação tem como base as propostas de Roma et al (2001) e de Galante et al.(2002), usando as seguintes características: versionamento de esquemas parcial; suporte à dimensão bitemporal; gerenciamento síncrono; utilização de invariantes nas requisições de evolução do esquema; e estratégia de armazenamento dos esquemas em múltiplos repositórios. Além disso, as características do TVM foram incorporadas para contemplar a evolução de esquemas segundo este modelo de dados.

5 TVM e TVQL

Neste capítulo são apresentados o Modelo Temporal de Versões (TVM) e a sua linguagem de consultas, a TVQL (*Temporal Versions Query Language*).

5.1 Modelo Temporal de Versões (TVM)

O Modelo Temporal de Versões (MORO ET AL., 2001-A; MORO, 2001) tem como base os conceitos de tempo e versão. Este modelo permite o armazenamento de versões de objeto, dos seus tempos de vida, e mantém o histórico das mudanças dos valores dinâmicos ou temporalizados. TVM é uma extensão do Modelo de Versões (GOLENDZINER, 1995) apresentado na seção 2.2.3.

No TVM, o tempo é associado ao objeto, à versão, aos atributos e aos relacionamentos. Um objeto possui uma linha de tempo para cada uma de suas versões. Desta forma, existem duas ordens de tempo em uso: (i) tempo ramificado para o objeto, (ii) tempo linear para cada versão. A variação temporal é discreta, e é representada no modelo pelo uso de rótulos temporais intervalares, que são bitemporais (tempo de transação e validade) e implícitos.

Os atributos e relacionamentos do objeto podem ser definidos como estáticos (quando o histórico dos valores não é armazenado) ou temporalizados (todas as alterações de seus valores são armazenadas formando seu histórico). A classificação de atributos e relacionamentos como temporalizados fica sob responsabilidade do usuário durante a especificação, sendo permitido que uma mesma classe tenha atributos de ambos os tipos (temporais ou não).

A figura 5.1 mostra a hierarquia de classes do TVM, a qual permite a definição de classes de aplicação como não temporais e não versionadas (alterações não são armazenadas), ou como temporais e versionadas (todas as alterações são armazenadas e o histórico é mantido). A classe *TemporalObject* possibilita a representação dos aspectos temporais, assumindo dois atributos: `start` (instante de criação do objeto) e `end` (instante da exclusão lógica do objeto). A variação dos atributos temporais e dos relacionamentos está associada aos rótulos predefinidos `vTimei`, `vTimef`, `tTimei`, `tTimef` que representam as validades inicial e final, e as transações inicial e final, respectivamente.

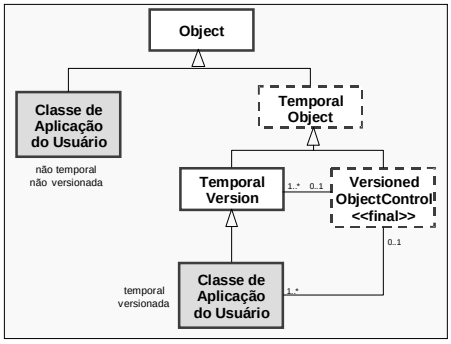


FIGURA 5.1 - Hierarquia de classes do TVM

As classes *TemporalObject* e *TemporalVersion* são abstratas, não podendo ser instanciadas diretamente. A classe *VersionedObjectControl* só pode ser instanciada pelo sistema gerenciador com o intuito de administrar os objetos que possuem versões. Essa classe é denominada classe final, não sendo permitido especializá-la em subclasses.

Quanto ao tempo de vida, o rótulo de tempo associado às versões deve estar contido no tempo de vida do objeto versionado, assim como o rótulo de tempo associado às variações dos atributos ou relacionamentos temporalizados de uma versão deve estar contido no tempo de vida da versão.

Os tempos de vida inicial e final do objeto são informados pelos valores de tempo de validade inicial e tempo de validade final do atributo `alive`. Essas regras de integridade valem dentro de cada "vida" do objeto. Por definição do Modelo, os tempos de vida de um objeto são elementos temporais. As vidas de um mesmo objeto não podem ter um (ou mais) instante(s) em comum. No momento em que um objeto excluído é restaurado, seu novo tempo de validade inicial tem que ser obrigatoriamente maior que o tempo de validade final anterior, nem que seja por um instante temporal.

Para manter a integridade dos rótulos temporais são consideradas as normas definidas por Hübler (2000), divididas em regras para inserção e atualização de dados. Qualquer informação que seja inserida em uma base de dados bitemporal deverá receber seus tempos de transação (fornecido pelo SGBD) e de validade (fornecido pelo usuário). O tempo de transação final é fornecido quando uma nova instância da informação é inserida, gerada por uma atualização da base de dados, ou quando é excluída. Quando o usuário não fornecer o tempo de validade final, esse será igual a *null* (valendo até que outra informação seja definida ou o objeto seja excluído).

A exclusão física é utilizada quando o usuário deseja remover informações que não são mais relevantes à aplicação. Esta operação é chamada *vacuuming* (RODDICK, 1994; 1996) e acontece raramente, somente quando o usuário deseja reduzir o volume dos dados armazenados. O TVM não define regras para este tipo de exclusão, assumindo que qualquer exclusão é executada logicamente. Além disso, a exclusão física restringe a possibilidade de retorno a qualquer estado passado do banco de dados. Se necessária, deve-se cuidar para evitar um impacto negativo na utilidade dos dados que permanecem na base (JENSEN, 1999; SNODGRASS, 2000).

5.1.1 Estados das versões

Para caracterizar o desenvolvimento de uma versão de objeto ou de sua consistência são utilizados estados, que são alterados quando ocorrem operações sobre as versões. A figura 5.2 ilustra os estados e as operações que geram transições entre eles.

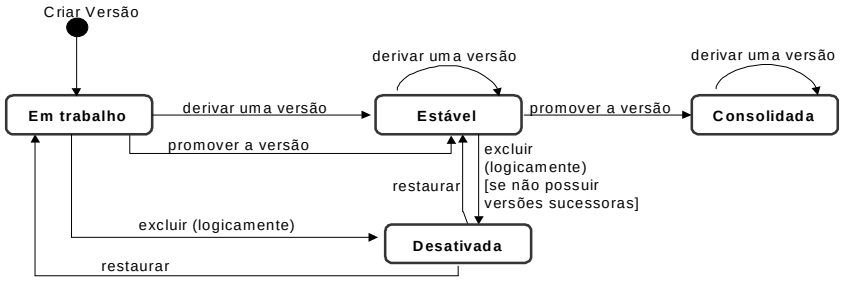


FIGURA 5.2 - Estados de uma versão

No momento da criação de uma versão, ela recebe o estado `em trabalho`. Neste estado a versão pode derivar uma nova versão, ser explicitamente promovida ou excluída logicamente. Quando a versão no estado `em trabalho` deriva uma nova versão ou é promovida, seu estado é alterado para `estável`. Caso ocorra uma exclusão lógica, seu estado passa para `desativada`.

Uma versão no estado `estável` pode derivar uma nova versão, pode ser excluída logicamente ou promovida. Quando deriva uma nova versão, a versão `estável`

permanece no mesmo estado; quando é promovida passa para o estado `consolidada`. Quando excluída logicamente, passa para o estado `desativada`. A exclusão lógica só pode ocorrer quando a versão não possui sucessoras.

Uma versão no estado `consolidada` pode servir de base para derivação, porém não pode ser excluída logicamente. Uma versão no estado `desativada` pode ser restaurada ao seu estado original.

Para que uma versão mude de estado são necessárias operações. A cada estado está associado um conjunto de operações possíveis, detalhadas a seguir.

5.1.2 Operações sobre versões

As operações sobre versões de objetos estão fortemente ligadas aos seus estados. A tabela 5.1 apresenta as operações e os estados das versões indicando quando podem ocorrer.

TABELA 5.1 - Estados das versões e respectivas operações

Estados / *Operações*	*Em trabalho*	*Estável*	*Consolidada*	*Desativada*
Derivar	*S*	*S*	*S*	-
Promover	*S*	*S*	-	-
Alterar	*S*	*N*	*N*	*N*
Excluir	*S*	*S (se não possuir sucessora)*	*N*	*N*
Consultar	*S*	*S*	*S*	*S*
Compartilhar	*N*	*S*	*S*	*N*
Restaurar	-	-	-	*S*

Legenda: ***S*** *pode ser realizada* ***N*** *não pode ser realizada* **–** *não definida*

A operação de exclusão sobre versões pode ser lógica ou física. Quando uma versão é excluída logicamente, ela recebe o estado de `desativada` e seu tempo de vida é encerrado. No momento da exclusão lógica, todos os atributos e relacionamentos temporais, se existirem, recebem o mesmo rótulo temporal final definido para um objeto ou versão. O TVM não oferece suporte à exclusão física. Uma versão no estado `desativada` pode ser restaurada. A restauração acontece para (MORO, 2001):

- uma versão folha que possui predecessora, isto é, não é a primeira versão do objeto;
- uma versão sem predecessora (primeira versão). Esse é um caso especial de restauração no qual o objeto não tem versões;
- uma versão cuja predecessora foi excluída;
- todas as versões do objeto. É possível restaurar todas as versões do objeto quando não se tenha gerado outra seqüência de derivação;
- um objeto versionado com versões.

5.1.3 Identificador de objetos - tvOID

A identificação das instâncias é realizada pela seqüência entidade, classe e versão, isto é, o `tvOID` que é definido na classe `Object`. Os objetos das classes comuns (sem tempo e sem versão) têm a mesma estrutura de `tvOID`, com a diferença de que o número da versão é sempre `null`.

Os objetos das classes temporais versionadas sempre começam com o número de versão igual a um (1). No momento de uma derivação, ocorrem dois processos:

- a criação do objeto versionado (número de versão é zero), que possui um relacionamento com a classe `VersionedObjectControl` (para a gerência do objeto e suas versões);
- a criação da nova versão em si e a atualização dos atributos de `VersionedObjectControl`.

5.1.4 Relacionamentos

O TVM é baseado no modelo orientado a objetos. Devido a esta característica, permite os seguintes tipos de relacionamento (MORO, 2001):

1) associação – a associação entre classes normais e temporal versionada é permitida desde que o relacionamento não seja temporal;
2) herança por refinamento – a herança por refinamento pode ser definida sem problemas entre duas classes normais. Entre duas classes temporais versionadas a herança obrigatoriamente deve ser por extensão, assim como uma classe temporal versionada não pode ter uma superclasse normal. Uma classe normal também não pode ter uma superclasse temporal versionada, pois isso implica diretamente no seu comportamento, devendo a classe se tornar temporal versionada;
3) herança por extensão – uma classe temporal versionada pode ter um ascendente em uma classe normal, sendo que a correspondência só pode ser 1:1 ou n:1, pois qualquer versão está associada a um único ascendente. Este relacionamento é detalhado na seção 5.1.5;
4) agregação – uma classe normal não pode ter uma (ou mais) classe temporal versionada como agregada, pois um objeto agregado pode ter várias versões e seria necessária a operação de configuração para resolver as referências dinâmicas. Em contrapartida, uma classe temporal versionada pode ter como componente uma (ou mais) classe normal, pois a operação de configuração obtém o único objeto definido para a entidade na classe.

5.1.5 Herança por extensão

A herança por extensão relaciona duas versões, dois objetos versionados ou uma versão e um objeto versionado, permitindo a modelagem em níveis de abstração diferentes, a visualização fácil e direta das diferenças entre os objetos, e o agrupamento das semelhanças das alternativas em versões. Além disso, inibe a proliferação excessiva de versões, pois são instanciadas somente aquelas realmente necessárias e nunca com valores nulos.

Na herança por extensão, a cardinalidade do relacionamento diz respeito às versões dos objetos e não aos objetos. Cada versão deve possuir pelo menos um ascendente que lhe corresponda, pois a criação de uma versão implica obrigatoriamente na ligação a um ascendente (a menos que seja raiz na hierarquia). Uma versão também pode apresentar mais de um ascendente, significando que as mesmas características definidas no seu nível podem ser ligadas a diferentes características no nível superior da hierarquia por extensão.

Pode ocorrer a existência de múltiplos ascendentes para um objeto (versionado, sem versões ou versão) em uma subclasse, caso o objeto ascendente possua versões. É permitido ao usuário estabelecer restrições de cardinalidade (mapeamentos) entre versões de um objeto. Essas restrições são chamadas correspondências e são modeladas como a cardinalidade do relacionamento de herança por extensão:

- 1:1 – cada versão na subclasse corresponde a exatamente uma versão na superclasse;
- 1:n – cada versão na subclasse pode corresponder a várias versões na superclasse, e várias versões na superclasse podem corresponder a somente uma versão na subclasse;
- n:1 – uma versão na subclasse pode corresponder a somente uma versão na superclasse, mas uma versão na superclasse pode corresponder a várias versões na subclasse;
- n:m – várias versões na subclasse podem estar relacionadas com uma versão na superclasse, e cada versão na subclasse pode corresponder a várias versões na superclasse.

As cardinalidades (1:0 e n:0) não existem nas correspondências, pois sempre um descendente deve estar relacionado a um ascendente. Caso a cardinalidade seja omitida, é assumido 1:1.

5.1.6 Configuração

A construção de uma configuração é solicitada para uma versão (chamada versão *base*) por meio da operação específica `getConfiguration`. Deve ser definida uma versão para cada ascendente na hierarquia por extensão e uma para cada componente das classes agregadas. Caso alguma classe agregada seja especificada não temporal versionada, o método retorna o `tvOID` correspondente à entidade requisitada. Podem ser necessárias várias escolhas para uma versão que apresenta uma ou mais referências dinâmicas e/ou múltiplos ascendentes em mais de uma superclasse. O processo de configuração é recursivo e consiste de dois passos para cada versão:

a) resolução de cada referência dinâmica em agregações, escolhendo uma das versões do objeto versionado referido;
b) escolha da versão ascendente para cada uma das superclasses.

Cada escolha define uma "parte" da configuração. Quando a configuração está completamente especificada, é gerada uma versão configurada, como sucessora de sua versão base. Uma versão configurada só pode fazer referência a versões configuradas, que podem ser compartilhadas por outras versões. Essa restrição garante que versões configuradas sejam sempre completamente definidas. Uma versão configurada é sempre criada a partir de uma versão existente, pemanecendo ligada a ela como sucessora, e possui o controle de sua vida por meio do atributo `alive`.

5.1.7 Exemplo de uma aplicação utilizando o TVM

Para exemplificar o uso do TVM é utilizado um resumo do estudo de caso apresentado em Moro (2001) e Zaupa (2002) que consiste na modelagem de uma empresa de criação de *websites*. Nessa empresa são armazenados os *sites* dos seus clientes e também são mantidas as páginas profissionais dos seus empregados. Um *website* é composto de uma ou mais páginas, sendo uma delas a inicial. Cada *website* possui um padrão de página associado composto pela cor e imagem de fundo, por um *banner*, e pelas especificações *default* da fonte. Esse padrão é usado como controle do *layout* da página de todos os empregados. Segundo especificação da empresa, esse padrão varia conforme as estações do ano e datas comemorativas. Por exemplo, no mês de aniversário da empresa, a imagem de fundo apresenta um bolo com velas acesas, e o *banner* mostra as ofertas especiais do mês.

Por determinação da empresa, todos os funcionários possuem uma página base constituída apenas de texto. Como alternativa, os funcionários podem possuir uma

página mais elaborada com imagens, mantendo as informações da versão base. Procurando se manter sempre atualizada, a empresa pode utilizar também páginas em XML para armazenar os dados dos seus funcionários.

A figura 5.3 apresenta uma parte do diagrama de classes do modelo. As especificações do cliente e dos empregados que possuem *websites* não são apresentadas. Também as operações estão omitidas em todas as classes.

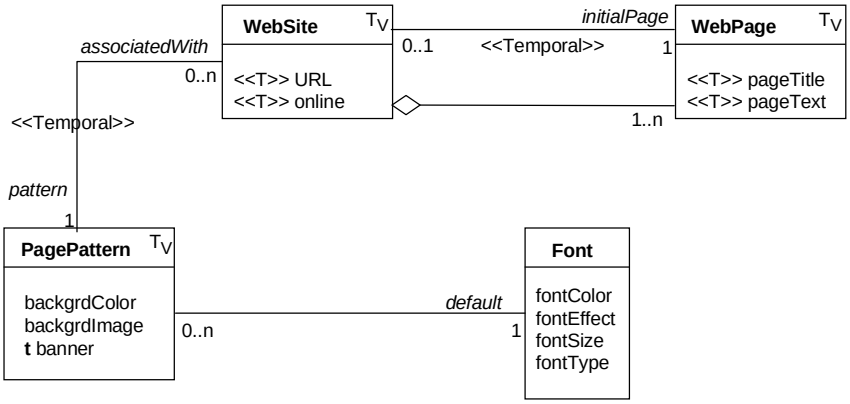
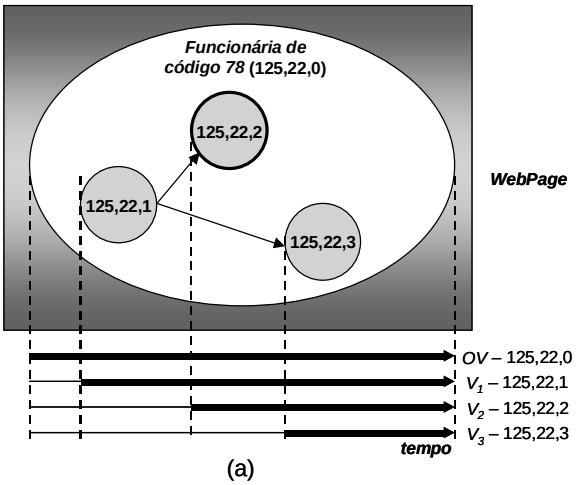


FIGURA 5.3 - Diagrama de classes do exemplo

As características de tempo e versão são utilizadas nas classes `WebSite`, `WebPage`, `PagePattern` e nos relacionamentos `pattern` (inverso `associatedWith`) e `initialPage`. O relacionamento `pattern` é considerado um relacionamento chave do modelo, pois permite a troca de padrão dos *websites* dos empregados para que seja atualizado de acordo com as estações do ano ou datas comemorativas.

Nesse contexto, a figura 5.4 apresenta a evolução das versões das páginas da funcionária de código 78, bem como o conteúdo de suas instâncias de forma simplificada. Essa funcionária possui sua página base ("`E78base.htm`") e sua página com imagens ("`E78gr.htm`"). No dia dezoito de março foi armazenado na base o nome de casada desta funcionária. Esta alteração gerou a atualização do título da página com imagens ("`Ciclana Xavier`") e a derivação de uma nova página base ("`E78base2.htm`"), pois o estado da página base original é `estável` e não pode sofrer alterações. A segunda versão desses objetos foi escolhida pelo usuário como a versão corrente.



(a)

tvoid	alive	configuração	estado	pageTitle	pageText
125,22,1	verdadeiro	falso	estável	Ciclana Saraiva	E78base.htm
125,22,2	verdadeiro	falso	em trabalho	Ciclana Xavier	E78gr.htm
125,22,3	verdadeiro	falso	em trabalho	Ciclana Xavier	E78base.htm

(b)

FIGURA 5.4 - (a)Representação gráfica das versões e das suas linhas de tempo
(b)Apresentação dos valores das instâncias da aplicação sem os tempos associados

No momento da consulta ao objeto versionado são retornados os valores da segunda versão. Obedecendo às regras de integridade temporal definidas em Moro (2001), quando o objeto versionado é instanciado (número de versão igual a zero), o seu tempo de validade inicial é definido como igual ao tempo de validade da primeira versão do objeto (número de versão igual a um).

Para exemplificar a forma de armazenamento dos atributos temporais do modelo, a tabela 5.2 apresenta a evolução dos estados das versões. Os demais atributos que armazenam informações como a "vida" de uma versão (`alive`), seus ascendentes e descendentes, predecessores e sucessores, entre outros não são ilustrados por possuírem o mesmo mecanismo para o armazenamento de suas respectivas evoluções.

TABELA 5.2 - Representação da evolução dos estados das versões do exemplo

tvoid	valor	TTIMEI	TTIMEF	VTIMEI	VTIMEF
125,22,1	em trabalho	12/02/2002	14/02/2002	12/02/2002	
125,22,1	em trabalho	15/02/2002		12/02/2002	14/02/2002
125,22,1	estável	15/02/2002		15/02/2002	
125,22,2	em trabalho	15/02/2002		15/02/2002	
125,22,3	em trabalho	18/03/2002		18/03/2002	

Os campos `TTIMEI`, `TTIMEF`, `VTIMEI` e `VTIMEF` da tabela 5.2 representam os tempos de transação inicial e final, e os tempos de validade inicial e final respectivamente. Um estado é dito atual se não possui valores armazenados nos tempos finais de transação e validade. Desta forma, é possível conhecer o momento de derivação da versão "`125,22,2`" a partir da versão "`125,22,1`". Esta operação gera uma alteração no estado da versão "`125,22,1`" que passa para `estável`. Uma nova derivação é realizada gerando a versão "`125,22,3`", porém esta operação não altera o estado da primeira versão.

5.2 Linguagem de Consulta para o Modelo Temporal de Versões – TVQL

A linguagem de consulta do Modelo Temporal de Versões (TVQL) (MORO ET AL., 2001-B; ZAUPA, 2002; MORO ET AL., 2002) é baseada na SQL (*Structured Query Language*), e mantém as características apresentadas na tabela 5.3.

TABELA 5.3 - Características gerais da TVQL

Estrutura base	`SELECT <lista de colunas, funções e predicados sobre colunas>`
	`FROM <lista de tabelas>`
	`[WHERE <condições de busca>]`

Alias	na cláusula `FROM`
Eliminação de duplicatas	com `SELECT DISTINCT`
Operadores	- relacionais: <, >, =, >=, <=, <> - lógicos booleanos: `OR`, `AND`, `NOT` - de conjuntos: `UNION`, `INTERSECTION`, `DIFFERENCE` - de condições compostas: `IN`, `BETWEEN AND` - de combinações de padrões: `LIKE` funções de agregação: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`
Cláusulas para grupo de dados	- `GROUP BY` - `HAVING` - `ORDER BY`

Para a consulta do histórico dos objetos foi definida a palavra reservada `EVER`. Sem ela, a consulta considera apenas os valores atuais dos dados dos objetos ativos, isto é, dos objetos não excluídos. `EVER` pode ser usada nas cláusulas `SELECT` e `WHERE`. Usando `EVER` na cláusula `SELECT`, a consulta retorna o histórico das propriedades temporais selecionadas. O histórico das propriedades temporais mencionadas é considerado quando `EVER` for usado na cláusula `WHERE`. O resultado da consulta com `EVER` depende dos atributos serem temporais. Para evitar repetições é necessário acrescentar a palavra reservada `DISTINCT`. Quando o usuário deseja a seleção dos valores atuais considerando uma cláusula `WHERE` temporal, é utilizada a palavra reservada `EVER` na cláusula `WHERE`.

Utilizando `EVER` são, portanto, considerados os históricos de todas as propriedades temporais mencionadas. Para anular esta condição é estabelecida a função `PRESENT`, que considera os valores atuais das propriedades na cláusula `WHERE` que estiverem dentro de sua expressão parâmetro.

Propriedades estabelecidas para a recuperação dos tempos de vida inicial e final dos objetos e versões, podem ser usadas nas cláusulas `SELECT` e `WHERE`. Os tempos de vida inicial e final de um objeto são equivalentes aos tempos de validade inicial e final do atributo `alive`, quando este possui valor `true`. Essas propriedades pré-definidas apenas deixam transparente para o usuário a implementação dos tempos de vida. A tabela 5.4 apresenta as condições temporais que podem ser usadas na TVQL.

TABELA 5.4 - Condições temporais na TVQL

tiInstant tfInstant viInstant vfInstant propriedade	= > < <= <>	Now	compara o instante com o instante atual
		valorTempo	compara o instante com o instante apresentado por ValorTempo
		propriedade	compara o instante com o valor da propriedade (definida com o tipo TIMESTAMP)
tinterval	INTERSECT	[valor1.. valor2]	compara o intervalo de transação ou validade com o intervalo definido entre os instantes valor1 e valor2
	OVERLAP	[.. valor]	compara o intervalo de transação ou validade com o intervalo definido de infinito passado até o instante valor
vinterval	EQUAL	tInterval/ vInterval	compara os intervalos pré-definidos entre eles
tiInstant tfInstant viInstant vfInstant propriedade	BEFORE INTO	[valor1.. valor2]	verifica se o instante ou o intervalo está antes, dentro ou após o intervalo definido entre os instantes valor1 e valor2
		[.. valor]	verifica se o instante ou intervalo está antes, dentro ou após o intervalo definido de infinito passado até o instante valor
		[valor ..]	verifica se o instante ou intervalo está antes, dentro ou após o intervalo definido a partir do instante valor até infinito futuro
tInterval	AFTER	tInterval	verifica se o instante ou intervalo está antes, dentro ou após o intervalo de transação
vInterval		vInterval	verifica se o instante ou o intervalo está antes, dentro ou após o intervalo de validade

5.2.1 Formas de consulta com seleção normal (SELECT propriedades)

A cláusula SELECT apresenta os nomes das propriedades que compõem o resultado com os dados atuais armazenados. Independentemente de que a propriedade seja temporal e que a cláusula WHERE considere aspectos temporais, os resultados retornados são os valores atuais da base. A cláusula FROM apresenta as classes a serem consideradas. A cláusula WHERE pode apresentar (ZAUPA, 2002):

- uma condição que considera apenas os dados atuais no estado atual da base, como por exemplo, a comparação entre uma propriedade e um valor específico;
- uma condição que considera o histórico dos dados mediante o uso da palavra reservada EVER, como por exemplo, a comparação entre os valores do histórico de uma propriedade e um valor específico;
- uma condição temporal que considera os rótulos temporais alterando o período de validade ou transação da base.

5.2.2 Formas de consulta com seleção temporal (SELECT EVER propriedades)

Nesse caso, a palavra reservada EVER segue a palavra SELECT, informando que é retornado o histórico das propriedades temporais da cláusula. A cláusula FROM apresenta as classes a serem consideradas. A cláusula WHERE pode apresentar (ZAUPA, 2002):

- uma condição que considera o histórico dos dados, como por exemplo, a comparação entre os valores do histórico de uma propriedade e um valor específico;
- uma condição que considera apenas os dados atuais no estado atual da base por meio do uso da palavra reservada `PRESENT`, como por exemplo, a comparação entre uma propriedade e um valor específico;
- uma condição temporal que considera os rótulos temporais alterando o período de validade ou transação da base.

5.2.3 Características de versionamento

Na cláusula `FROM`, os objetos e versões a serem consultados devem ser definidos pelo usuário. Os objetos e versões correntes são escolhidos pelo nome da classe à qual pertencem. Todas as versões dos objetos são consideradas, acrescentando na cláusula `FROM` o nome da classe seguido da palavra reservada `versions`. Se a cláusula `FROM` contiver apenas o nome da classe, retorna somente as versões correntes dos objetos.

Esta seção apresenta as funções e propriedades específicas para a recuperação das informações de versões e objetos versionados. As funções com o sufixo `at` retornam o valor da informação relativo a um determinado ponto no tempo de validade (ZAUPA, 2002).

Funções que recuperam o estado:

- `isWorking – boolean;`
- `isWorkingAt (instant) – boolean;`
- `isStable – boolean;`
- `isStableAt (instant) – boolean;`
- `isConsolidated – boolean;`
- `isConsolidatedAt (instant) – boolean;`
- `isDeactivated – boolean;`
- `isDeactivatedAt (instant) – boolean.`

Funções da navegação na hierarquia de herança:

- `isAscendantOf (descTVObject) – boolean;`
- `isDescendantOf (ascTVObject) – boolean.`

Funções da navegação na hierarquia de derivação:

- `isFirst – boolean;`
- `isFirstAt (instant) – boolean;`
- `isLast – boolean;`
- `isLastAt (instant) – boolean;`
- `isSuccessorOf (predTVObejct) – boolean;`
- `isSuccessorOfAt (predTVObject, instant) – boolean;`
- `isPredecessorOF (succTVObject) – boolean;`
- `isCurrent – boolean;`
- `isCurrentAt (instant) – boolean;`
- `isConfiguration – boolean.`

Além das funções, é definido um conjunto de propriedades para que o controle do objeto versionado possa ser consultado de maneira transparente. Essas propriedades podem ser recuperadas na cláusula `SELECT` ou participa de condições na cláusula `WHERE` (ZAUPA, 2002):

- `configurationCount` – número de versões configuradas;
- `currentVersion` – `tvOid` da versão corrente;
- `firstVersion` – `tvOid` da primeira versão;
- `lastVersion` – `tvOid` da última versão;
- `nextVersionNumber` – número da próxima versão;
- `userCurrentFlag` – valor booleano para o *flag* da versão corrente especificada pelo usuário;
- `versionCount` – número de versões do objeto versionado.

5.3 Considerações finais

Este capítulo abordou as principais características do TVM, modelo utilizado no protótipo implementado. Foram apresentados a sua hierarquia, os estados dos objetos, sua definição de classes, atributos, relacionamentos e configurações. Também foram apresentadas a estrutura da TVQL, suas formas de consulta e as funções para recuperação do histórico e de versões de objeto. Mais detalhes sobre o TVM e sua linguagem de consulta TVQL podem ser encontrados em Moro (2001); Moro et al. (2001-a, 2001-b, 2002), e Zaupa (2002). As consultas sobre dados no sistema implementado são feitas utilizando uma extensão desta TVQL para considerar também a evolução dos esquemas, apresentada na seção 6.2.6.

No próximo capítulo é feita a análise do versionamento de esquemas considerando o TVM como modelo de dados. São apresentadas características como os estados de uma versão de esquema, as operações sobre eles, além da proposta de uma arquitetura para implementar o versionamento de esquemas sobre o referido modelo.

6 Implementação

Nos capítulos anteriores foram apresentadas as características necessárias para que ocorra o versionamento de esquemas em um ambiente temporal e o TVM. Neste capítulo será abordado o versionamento de esquemas sobre o TVM detalhando suas características. Nesta análise encontram-se as definições utilizadas no protótipo implementado.

6.1 Arquitetura proposta para o versionamento de esquemas sobre o TVM

Definir uma arquitetura de um SGBD para a implementação de um modelo de dados com suporte a tempo é uma tarefa de alto custo (JENSEN, 1999). Este investimento aumenta ainda mais quando se acrescenta o suporte a versões. Para viabilizar este propósito, optou-se por definir uma camada com características de tempo e versão para realizar o suporte do versionamento de esquemas sobre um SGBD existente (figura 6.1).

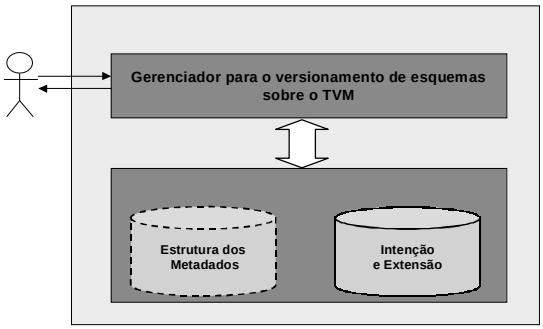


FIGURA 6.1 - Arquitetura do gerenciador para o versionamento de esquemas

Tendo como meta a implementação, é necessário definir as funcionalidades necessárias ao gerenciador para o versionamento de esquemas. Com esta finalidade, foi adaptada a idéia apresentada em Moro (2001), acrescentando o acesso ao repositório do metabanco, e aos repositórios das versões de esquema e as suas instâncias (figura 6.2).

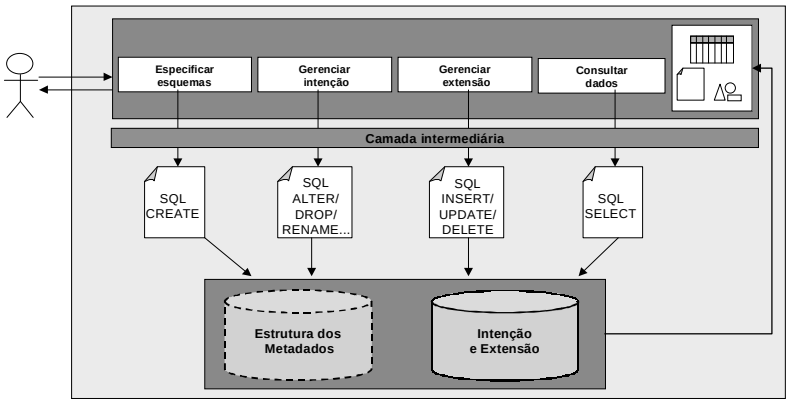


FIGURA 6.2 - Funcionalidades do gerenciador para o versionamento de esquemas

O gerenciador para o versionamento de esquemas possui os seguintes módulos:

- Especificar esquemas – responsável pela criação e correção de esquemas;
- Gerenciar intenção – responsável pelo controle das versões de esquema;
- Gerenciar extensão – responsável pela administração dos dados armazenados em seus respectivos repositórios;

- Consultar dados – responsável por consultas ao repositório dos metadados e aos repositórios das versões de esquema.

Na camada intermediária é realizada a verificação das invariantes definidas no sistema (seção 6.1.1) para tempo e versões, a fim de manter a base sempre correta e consistente.

6.1.1 Invariantes do sistema

Para a correta evolução de esquemas com uso de versões em um ambiente temporal é necessário definir um conjunto de regras de integridade. Estas, também chamadas de invariantes, são verificadas toda vez que ocorre uma operação capaz de gerar uma nova versão. Com base nas propostas de Edelweiss, Castilho e Oliveira (1995), Golendziner (1995), Liu (1997), Galante (1998) e Mandreoli (2001), o sistema implementado utiliza as seguintes invariantes:

- unicidade de nomes – os nomes dos esquemas, das classes, de suas propriedades e métodos devem ser únicos. Se a classe é definida como subclasse, herda as propriedades de sua superclasse e as novas propriedades definidas devem ter nomes únicos;
- todas as propriedades devem ter um domínio definido;
- se uma classe é definida como subclasse, a superclasse correspondente precisa ser definida;
- todos os objetos referidos devem estar presentes no banco de dados;
- os valores definidos para todos os atributos de uma versão de instância devem pertencer ao domínio definido para a versão da classe a qual está associado ou igual a `NULL`.

Para a definição do gerenciamento temporal de versões para evolução de esquemas, quando o TVM é empregado, faz-se necessária também a definição de restrições de integridade temporais. Os rótulos temporais dos elementos do esquema (atributos, classes, relacionamentos, métodos e parâmetros) devem estar contidos no intervalo temporal (tempo de vida) da versão do esquema a que pertencem. O tempo de vida de uma versão de esquema é o intervalo compreendido entre os seus tempos de transação e/ou validade iniciais e finais. No protótipo implementado os esquemas são de tempo de transação e os dados são TVM. As regras temporais para os objetos são as mesmas definidas em Moro (2001), portanto não serão aqui apresentadas. As regras de integridade temporal utilizadas nas versões de esquema são as seguintes:

- o tempo de transação inicial de cada elemento do esquema deve ser maior ou igual ao tempo de transação inicial do esquema, e menor ou igual ao tempo de transação final do esquema ao qual pertence;
- o tempo de transação final de cada elemento do esquema deve ser maior ou igual ao tempo de transação inicial do esquema, e menor ou igual ao tempo de transação final do esquema ao qual pertence;
- o intervalo de transação de uma classe deve estar contido no intervalo de transação da versão do esquema ao qual pertence;
- o intervalo de transação de uma propriedade (como atributos e métodos apresentam as mesmas restrições de integridade, ambos são tratados como propriedades), deve estar contido no intervalo de transação da classe a que pertence;
- o intervalo de transação de um relacionamento deve estar contido no intervalo de transação das classes envolvidas.

Esses critérios foram inseridos para garantir que cada operação de atualização preserve a integridade do esquema e a validade do banco de dados. A derivação de versões deve sempre resultar em esquemas estruturalmente válidos e corretos. Desta forma, modificações são somente realizadas se garantirem esta consistência, caso contrário, devem ser recusadas e o usuário notificado.

6.2 Especificações do protótipo implementado

O objetivo deste trabalho é a implementação do versionamento de esquemas baseado na arquitetura proposta na seção 6.1. Para a sua realização foram definidos controles que permitem a análise da evolução dos esquemas como um todo e, para cada esquema, as correspondentes versões de seus dados e a sua evolução temporal.

O protótipo implementado para o gerenciamento do versionamento de esquemas sobre o TVM, o TVMSE (*Temporal Versions Model Schema Evolution*), possui as seguintes características:

- versionamento de esquemas – tempo de transação;
- versionamento dos dados – TVM (bitemporal);
- armazenamento – múltiplos repositórios;
- gerenciamento – síncrono;
- alternativa de derivação – alternativa 1 (seção 3.3.1).

No TVMSE o gerenciamento ocorre por meio de três níveis que caracterizam uma estratégia para o versionamento de esquemas sobre o TVM (figura 6.3).

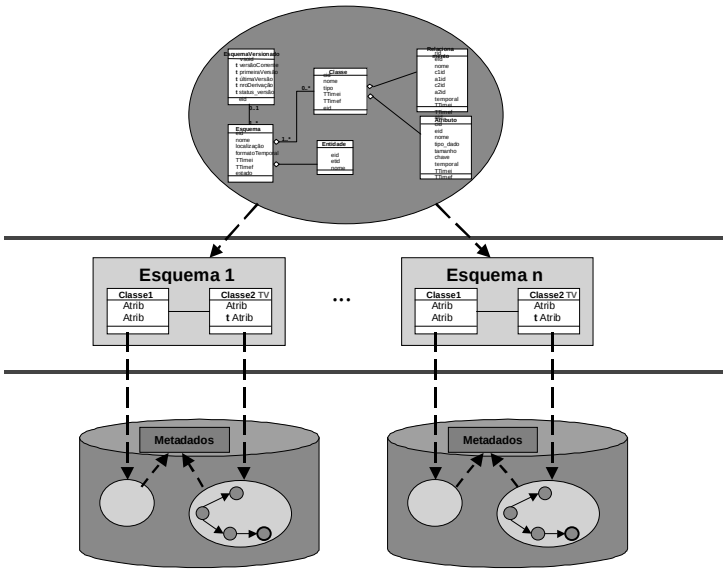


FIGURA 6.3 - Estratégia para o gerenciamento do versionamento de esquemas

No primeiro nível, encontra-se o meta-esquema que gerencia todas as versões de esquema. No segundo, encontram-se as especificações de cada versão de esquema, e no terceiro nível encontram-se os repositórios de cada versão de esquema. Dentro de cada repositório encontra-se uma estrutura denominada metadados que realiza o controle da evolução das instâncias TVM. Estas e as demais características da implementação serão apresentadas e justificadas com maiores detalhes nas subseções seguintes deste capítulo.

6.2.1 Gerenciamento da intenção

As versões de esquema são de tempo de transação, isto é, estas são armazenadas em um metabanco de tempo de transação, onde somente esta dimensão temporal é suportada. Como este trabalho foi um primeiro esforço de implementação para demonstrar o versionamento de esquemas sobre o TVM optou-se por utilizar somente uma dimensão temporal na intenção, pois na extensão é necessário efetuar o controle das duas dimensões temporais (tempo de transação e de validade).

O esquema é armazenado em um único repositório, chamado metabanco, com múltiplos esquemas. Esta alternativa foi escolhida por tornar mais simples a recuperação das informações das versões de esquemas quando estas são armazenadas fisicamente em um mesmo local. Os componentes das versões de esquemas são armazenados de acordo com as especificações do meta-esquema no metabanco.

Diagrama de estados

A alternativa 1 (seção 3.3.1) foi escolhida para representar a forma de derivação das versões de esquema. Esta escolha foi realizada devido a sua simplicidade no que diz respeito ao gerenciamento, depois de efetuar a implementação desta alternativa torna-se possível extendê-la para contemplar as demais. De acordo com esta escolha, as modificações no esquema são sempre realizadas sobre o esquema corrente, que é o último, não sendo permitidas alterações nos esquemas passados. A partir do diagrama de estados do TVM, apresentado na figura 4.1 (seção 4.5), os estados e as operações sobre estes estados são a seguir reformulados para o caso específico desta alternativa de implementação.

Durante o seu desenvolvimento, uma versão de esquema pode assumir três estados, são eles: `em trabalho`, `estável` e `congelada`. O estado `consolidada` foi suprimido, pois na alternativa escolhida não se percebe a necessidade de um estado em que é proibida a modificação tanto da versão de esquema como da sua base, sendo este controle realizado diretamente pela aplicação desenvolvida. A figura 6.4 apresenta o diagrama de estados quando é utilizada a alternativa 1 como forma de derivação das versões de esquema.

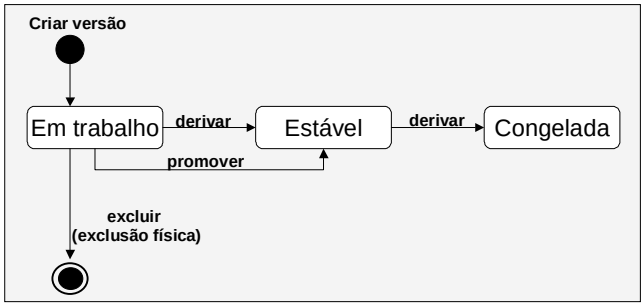


FIGURA 6.4 - Diagrama de estados para a alternativa escolhida

Uma versão de esquema no estado `em trabalho` é uma versão temporária, não possui tempos associados, operações de modificação geram correções e pode ser excluída fisicamente.

Uma versão de esquema `estável`, é uma versão que não pode mais ser excluída fisicamente, possui tempos associados e as operações de modificação geram a derivação de uma nova versão. Para evitar a replicação de versões de esquema que não estão completamente prontas, e assim evitar a criação desnecessária de repositórios optou-se por diferenciar quando uma versão de esquema `estável` pode ser instanciada ou não, evitando desta forma o desperdício do meio físico de armazenamento. Para que

ocorra a instanciação é necessário antes criar fisicamente o repositório da versão de esquema. Assim, existem dois tipos de versão de esquema `estável` que são `não ativa` e `ativa`. No momento em que uma versão `estável` é ativada pelo usuário, ela pode ter instâncias, pois seu repositório foi criado na base. Somente versões estáveis podem ser consideradas como a versão corrente. Durante a evolução de uma versão de esquema, enquanto esta não puder ser instanciada, a versão corrente é sempre a última derivada. No instante em que uma versão de esquema tem uma base associada, ela passa a ser a corrente até que uma nova versão derivada se torne apta para receber instâncias. Quando ocorre este processo, a versão derivada passa a ser a corrente.

Uma versão de esquema no estado `congelada`, é uma versão que teve seu tempo de vida encerrado, estando disponível somente para consultas.

Toda vez que uma operação de modificação na estrutura do esquema é realizada, são requisitadas operações sobre os estados das versões. A seguir são apresentadas as operações utilizadas na implementação, e sobre quais estados elas atuam (tabela 6.1).

TABELA 6.1 - Operações sobre os estados das versões de esquema da implementação

Estados / Operações	Em trabalho	Estável		Congelada
		Não ativa	ativa	
Instanciar			X	
Consultar	X	X	X	X
Alterar	X			
Derivar	X	X	X	
Excluir	X			

A escolha da alternativa 1 para a derivação define, também, o tipo de versionamento do esquema como parcial. Neste tipo de versionamento é possível alterar somente as versões atual e futura do esquema, não sendo permitidas alterações em versões passadas.

Meta-esquema

Para melhor representar o versionamento de esquemas utilizando o tempo de transação sobre o TVM, optou-se por remodelar o meta-esquema proposto em Galante et al (2002). O meta-esquema da implementação é composto das seguintes metaclasses: `Esquema`, `Entidade`, `Classe`, `Atributo`, `Relacionamento` e `EsquemaVersionado`. A figura 6.5 apresenta o meta-esquema utilizado na implementação. É importante salientar que o meta-esquema foi projetado para ser implementado em um banco de dados relacional no qual foram incluídos campos chave nas classes para a sua construção, e a inclusão dos rótulos temporais em cada uma destas. O *script* utilizado na criação do meta-esquema no banco de dados DB2 encontra-se no Anexo 1.

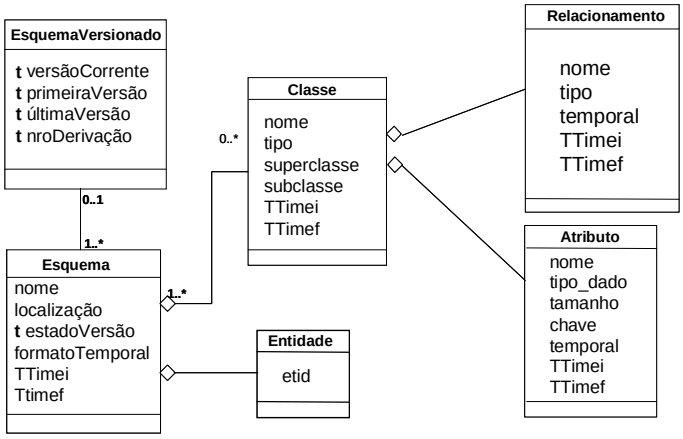


FIGURA 6.5 - Meta-esquema para o versionamento de esquemas

A metaclasse `Esquema` possui o identificador do esquema, seu nome, a indicação de sua localização, o formato temporal suportado, o identificador do esquema versionado, tempos de transação inicial e final (para representar o tempo de vida de um esquema), e o estado do esquema (este atributo é temporal, ou seja, permite o armazenamento tanto do estado atual quanto dos passados).

A metaclasse `Entidade` possui o identificador da entidade e o identificador do esquema correspondente.

A metaclasse `Classe` possui o indentificador da classe, o identificador do esquema à qual esta pertence, seu nome, seu tipo (normal ou temporal versionada), as identificações quanto a ser superclasse ou subclasse, e os seus tempos de transação inicial e final.

A classe `Atributo` possui o identificador do atributo, o identificador da classe à qual ele pertence, o identificador do esquema, o nome, o tipo de dado, se o atributo é temporal ou não, o tamanho do atributo (depende do tipo de dado), se é chave ou não, e seus tempos de transação inicial e final.

A classe `Relacionamento` possui o identificador do relacionamento, o identificador do esquema, seu nome, o identificador da classe1, o identificador do atributo de ligação na classe1, o identificador da classe2, o identificador do atributo de ligação na classe2, o tipo do relacionamento (se é uma associação, uma agregação, uma herança por refinamento ou por extensão), se o relacionamento é temporal ou não, e seus tempos de transação inicial e final.

A classe `EsquemaVersionado` possui o identificador do esquema versionado, a versão corrente, a primeira versão, a última versão, e o número de derivações. Com exceção dos identificadores, todos os atributos mantêm, além dos valores atuais, os seus históricos.

Mapeamentos da Intenção

No TVM é possível descrever os comportamentos estáticos e dinâmicos da aplicação. Para implementar estas características em um banco de dados convencional, é necessário que os rótulos temporais implícitos no modelo sejam mapeados de forma explícita, através da utilização de atributos.

Todas as classes sem tempo e sem versão da aplicação são mapeadas para tabelas com o mesmo nome da classe (denominadas tabelas principais) nas quais:

- cada atributo do meta-esquema é mapeado para uma coluna na tabela com os respectivos nome e tipo;

- mapeamento dos tipos de dados (tabela 6.2):

TABELA 6.2 - Mapeamento dos tipos de dados

TVM	DB2
`Integer`	`Integer`
`Real`	`Real`
`String`	`Varchar`
`Char`	`Char`
`Boolean`	`Char (1) CONSTRAINT cname CHECK (NomeColuna IN ('T', 'F'))`
`Date`	`Date`
`Time`	`Time`
`Instant`	`Timestamp`
`TVOID`	`Varchar (20)`
`Set`	-

- todas as classes recebem o atributo `tvoid` do tipo `varchar` como chave primária.

Para a representação das tabelas temporais versionadas que possuem atributos temporais, tornou-se necessário construir uma tabela para cada um dos atributos temporais. A cada um destes atributos é associado um intervalo de tempo de transação, representado pelos atributos de início e fim do respectivo intervalo temporal.

Armazenamento do Modelo Conceitual Temporal

O modelo conceitual temporal será armazenado em um metabanco de dados por intermédio das metaclasses, acrescidas de novos atributos necessários à representação. A granularidade temporal assumida é o "minuto", assim a composição dos tempos de transação e validade terá o formato "DD/MM/AAAAHHMM", onde:

- DD – representa o dia;
- MM – representa o mês;
- AAAA – representa o ano;
- HH – representa a hora;
- MM – representa o minuto.

Por optarmos implementar o versionamento dos esquemas somente através do tempo de transação, nas metaclasses não são inseridos os rótulos temporais de tempo de validade.

6.2.2 Gerenciamento da extensão

As instâncias TVM, além de apresentarem versões dos dados, são bitemporais. Optou-se pela implementação de múltiplos repositórios para implementar o versionamento de esquemas. Assim, cada versão do esquema conceitual possui o seu repositório específico, o que facilita a recuperação dos dados que são devidamente formatados de acordo com a versão à qual pertencem. As instâncias são gerenciadas por uma estrutura denominada metadados, presente em cada repositório de dados.

Metadados para o versionamento das instâncias

O mapeamento do TVM para um banco relacional foi definido em Moro et al. (2002), Zaupa (2002). Os dados da extensão são definidos de acordo com a hierarquia de classes do TVM (figura 6.6), onde somente a classe `VersionedObjectControl` necessita de mapeamento direto. Esta é mapeada para a tabela `VOC` a qual gerencia todos os objetos versionados do modelo. Para completar o

mapeamento das classes do TVM, são criadas as tabelas `PredSucc` e `AscDesc`, pois o DB2 não possui o tipo de conjunto. Para cada atributo definido na tabela `VOC` existe uma tabela auxiliar correspondente, pois são temporais.

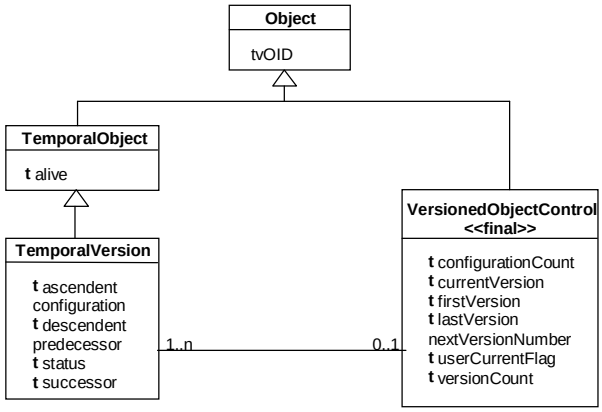


FIGURA 6.6 - Hierarquia de classes do TVM com atributos

As classes temporais versionadas são mapeadas para tabelas principais como tabelas normais[3]. No entanto, herdam os atributos da hierarquia base do TVM para suportar os aspectos temporais e de versões, bem como outras características particulares:

- os atributos herdados `alive`, `configuration` e `status` são mapeados para colunas na tabela principal;
- os atributos herdados `ascendant`, `descendant`, `predecessor` e `successor` são mapeados para duas tabelas especiais chamadas `AscDesc` e `PredSucc`, que armazenam os pares e rótulos temporais;
- toda tabela principal possui uma referência (chave estrangeira) chamada `refVOC`, que referencia o identificador (`tvoid`) da tabela de controle de objetos versionados (tabela `VOC`), para todos os objetos que possuam mais de uma versão.

Mapeamentos da extensão

Os relacionamentos de agregação/composição e herança são mapeados para relacionamento normal, no qual a classe agregada ou subclasse recebe a chave primária da classe principal como chave estrangeira.

Os atributos e relacionamentos temporais, definidos nas classes temporais versionadas, são mapeados em tabelas específicas denominadas tabelas auxiliares, as quais armazenam seus históricos. Estas possuem as seguintes características:

- o nome da tabela auxiliar é composto do nome da classe seguido do nome da propriedade;
- a estrutura da tabela auxiliar é sempre a mesma, sendo composta pelo `tvoid` que referencia o identificador da classe principal (chave estrangeira), pelo atributo `value` que é definido com o mesmo tipo definido para a propriedade, bem como pelos rótulos temporais `TTIMEI`, `TTIMEF`, `VTIMEI`, `VTIMEF` que representam os tempos de transação inicial e final, e tempos de validade inicial e final, respectivamente;
- o relacionamento entre a tabela auxiliar e a principal é de `1:n` (uma tabela principal para `n` auxiliares);

[3] O estudo de caso apresenta as tabelas criadas em cada repositório para o gerenciamento das instâncias.

- a chave primária da tabela auxiliar é composta pelo identificador (`tvoid`) e os tempos iniciais de transação (`TTIMEI`) e validade (`VTIMEI`).

Nas tabelas principais ficam armazenados somente os valores que não são temporais. Por outro lado, o armazenamento dos valores históricos e atuais é feito nas tabelas auxiliares para os atributos temporalizados.

Para o controle do versionamento dos dados da extensão foi implementada uma estrutura denominada metadados[4] (figura 6.7), conforme definição realizada em Zaupa (2002). Esta estrutura é inserida em todos os repositórios das versões de esquemas.

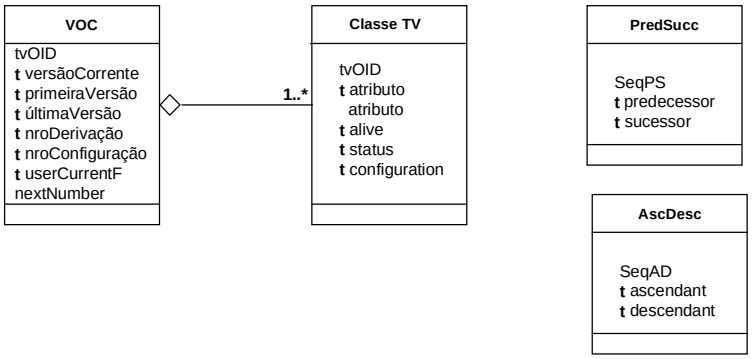


FIGURA 6.7 - Metadados

A cada novo valor de um atributo temporal, deve-se encerrar os tempos de transação e validade da tupla atual, e inserir uma nova tupla com os tempos finais de transação e validade com o conteúdo `null` – o final da validade poderá ser previamente definido, pois para o tempo de validade é permitido o lançamento de valores futuros. A tupla atual sempre será aquela que possuir o conteúdo `null` nos seus tempos de transação e/ou validade, ou aquela cujo tempo de validade está dentro do período referenciado.

Essa forma de mapeamento consiste na criação de uma tabela para cada classe da aplicação, sendo criada uma tabela auxiliar para cada atributo temporal presente nas classes a serem mapeadas. A tabela auxiliar armazena o `tvoid` da tabela principal, o valor histórico do atributo e os rótulos temporais. Para facilitar o gerenciamento, seu nome é formado pelo nome da tabela principal mais o nome do atributo temporal.

A eliminação física da tupla não é permitida. O que deverá ocorrer é o encerramento de sua validade, através da atualização dos atributos temporais de tempo de transação e/ou validade. Se a tabela possuir atributos temporalizados, esta atualização deverá ser propagada também para as tabelas auxiliares, geradas a partir dos atributos temporais pertencentes à tabela principal.

6.2.3 Sincronismo entre esquemas e dados

O gerenciamento é síncrono para os tempos de transação e de validade. É exigido que o esquema conceitual e os dados da extensão possuam a mesma pertinência temporal, isto significa que os dados podem ser recuperados e atualizados somente através da sua versão de esquema correspondente. Esta alternativa foi escolhida por tornar mais "natural" a recuperação dos dados da extensão de acordo com seu esquema conceitual, o que não ocorre no gerenciamento assíncrono, pois seria necessária a

[4] O *script* de criação dos metadados encontra-se no anexo 2.

implementação de vários mecanismos para obter as informações de um repositório segundo uma outra versão de esquema.

6.2.4 Estratégia de migração dos dados

A estratégia de migração dos dados está diretamente ligada à estratégia para a adaptação de instâncias (GOLENDZINER, 1995; LIU, 1997; WEI, ELMASRI, 2000; LERNER, 2000; ANGONESE, 2000; MANDREOLI, 2001; GALANTE ET AL., 2002). A estratégia utilizada neste trabalho é a de migração imediata, isto é, todas as tuplas devem ser copiadas do repositório atual para o novo, devidamente adaptadas ao novo esquema (adaptação das instâncias). Este processo deve ocorrer no momento em que o usuário permite a instanciação de uma versão de esquema.

As versões de objetos podem ser criadas gradativamente pelo usuário no repositório da versão de esquema que está `estável` e `ativa`, seguindo os requisitos definidos pelo Modelo Temporal de Versões (TVM), ou geradas automaticamente pelo sistema como forma de adaptar as instâncias à nova versão de esquema derivada recebendo os intervalos de transação e validade da nova versão. As instâncias que são copiadas para o novo repositório são aquelas que possuem o mesmo tempo de transação final da versão de esquema à qual pertenciam, ou seja, o tempo de transação final dos objetos é igual ao tempo de transação final da sua versão de esquema. Esta estratégia foi adotada para que não ocorra replicação desnecessária de objetos, ou seja, o histórico deles não é copiado, são copiados somente os valores atuais do repositório origem. O processo de migração dos dados para o novo repositório deve ser *on-line*, de forma a causar o menor impacto possível sobre as atividades normais do usuário. O sistema deve disponibilizar os dados do usuário somente para leitura, bloqueando os acessos de atualização até o término da migração dos dados.

6.2.5 Operações para o versionamento de esquemas no TVMSE

O protótipo implementado permite as seguintes mudanças que geram o versionamento de esquemas:

1) no esquema – inclusão ou exclusão de uma classe (exclusões em superclasses não são possíveis enquanto houver subclasses associadas); alteração no nome do esquema. Estas operações não geram uma nova versão de esquema quando o estado do esquema é `em trabalho`, sendo que esta observação se aplica às demais operações;
2) na classe – inclusão ou exclusão de um atributo; alteração no nome da classe; alteração no tipo da classe (de temporal versionada para normal se a classe não possuir nenhum atributo temporal);
3) no atributo – alteração no tipo de dado numérico (o TVMSE permite a transformação de inteiro para real, de real para decimal e vice-versa); alteração à menor no tamanho do atributo, se este for `string` ou `varchar`, e alteração no número de casas decimais; alteração no nome; alteração no tipo (temporal ou normal);
4) no relacionamento – alteração no(s) atributo(s) de ligação; alteração no nome do relacionamento.

Na base, as alterações em atributos sem tempo e sem versão geram a inserção de uma nova tupla na tabela principal. As alterações em atributos temporais geram a inclusão de uma nova tupla na tabela auxiliar e o encerramento dos tempos de transação e de validade respectivamente.

6.2.6 Consultas

Na literatura podemos encontrar vários tipos de linguagem de consulta para bancos de dados temporais (CHOMICKI, 1995; RODDICK, SNODGRASS, 1995; BÖHLEN, JENSEN, SNODGRASS, 2000; CHOMICKI, TOMAN, BÖHLEN, 2001), porém é raro encontrar uma linguagem de consulta que leve em consideração o versionamento de esquemas. No TVMSE são utilizadas duas estratégias para as consultas, uma para a intenção e outra para a extensão. As consultas ao metabanco (consultas às versões do esquema) utilizam a MSQL (GRANDI, 2002), e dentro de cada repositório (consultas aos dados) é utilizada a TVQL apresentada na seção 5.2.

Intenção

A fim de representar adequadamente a busca de informações em um ambiente de versionamento de esquemas onde as versões estão armazenadas em múltiplos repositórios, o protótipo utiliza a linguagem MSQL (*Multi-Schema Query Language*) proposta por Fabio Grandi (GRANDI, 2002). Esta precisou de uma adaptação para contemplar as consultas com aspectos de tempo. Para isto acrescentou-se a cláusula `ever` da TVQL para as consultas à intenção. Com esta adaptação é possível recuperar informações históricas no repositório do meta-esquema.

A sintaxe básica da MSQL, que estende a SQL, permite uma maneira de contextualizar nomes e referências aos dados para as versões de esquema. Em particular, utiliza-se a sintaxe "`[SV:X]`" para referenciar a entidade conceitual (tabela ou atributo) de nome "`X`" na versão de esquema "`SV`", e a sintaxe "`SV:X`"para referenciar a extensão da versão de esquema "`SV`" da entidade conceitual "`X`".

Para exemplificar a MSQL, temos as seguintes consultas:

1) `select * from [SVj:R]` – retorna as informações da intenção da tabela `R` segundo a versão de esquema `SVj`;
2) `select * from SVi:R` – retorna as informações da extensão da tabela `R` segundo a versão de esquema `SVi`;
3) `select [SVj:A] from R` – retorna as informações da intenção do atributo `A` da tabela `R` na versão de esquema `SVj`;
4) `select SVi:A from R` – retorna os valores da extensão do atributo `A` da tabela `R` na versão de esquema `SVi`;
5) `select ever [SVi: A] from R` – retorna o histórico das informações da intenção do atributo temporal `A` da tabela `R` na versão de esquema `SVi`;
6) `select ever SVj: A from R` – retorna o histórico das informações da extensão do atributo temporal `A` da tabela `R` na versão de esquema `SVj`.

A MSQL permite a generalização de consultas por intermédio das cláusulas `select` e `from`. Um exemplo desta generalização é a possibilidade de consultas assíncronas, isto é, recuperar dados de um determinado repositório de acordo com as especificações de outra versão de esquema. Embora este tipo de consulta seja possível na MSQL, no protótipo implementado não é possível a sua realização, pois optou-se por implementar o gerenciamento síncrono. Assim, os dados são recuperados e atualizados de acordo com a especificação do esquema no qual foram definidos.

Extensão

Os dados são armazenados no repositório de cada versão de esquema segundo o modelo TVM. Para o seu controle é utilizada a estrutura dos metadados. Assim, a linguagem que melhor atende as consultas à base é a TVQL.

A fim de adaptar a TVQL a um ambiente com versionamento de esquemas acrescentou-se a ela a cláusula `SET SCHEMA`, que no protótipo implementado indica qual

repositório deve ser utilizado para realizar a consulta. Para exemplificar esta adaptação à TVQL, temos as seguintes consultas:

1) `set schema SVj`
   `select ever a from R` – seleciona o repositório da versão de esquema `SVj`, retorna o histórico do atributo a da tabela temporal versionada `R`;
2) `set schema SVi`
   `select *`
   `from predsucc`
   `where predsucc.successor='1,1,2'` – seleciona o repositório do esquema `SVi`, retorna as versões que são sucessoras da versão de `tvoid='1,1,2';`
3) `set schema SVj`
   `select a`
   `from R.VERSIONS` – seleciona o repositório do esquema `SVj`, retorna as versões do atributo `a` da tabela versionada `R`.

Esta extensão permite fazer consultas aos dados do repositório de uma versão de esquema. Consultas que envolvam vários esquemas não são contempladas, nem a possibilidade de consultar vários repositórios numa mesma consulta.

6.3 Considerações finais

Neste capítulo foram apresentadas a arquitetura utilizada para o desenvolvimento do protótipo implementado (TVMSE) e as funcionalidades que este deve oferecer. Além disso, foram apresentados o diagrama de estados, as operações utilizadas no versionamento de esquemas e as invariantes utilizadas no protótipo implementado. A definição de invariantes visa garantir a validade do esquema frente às modificações realizadas. A cada operação de modificação esse mecanismo é acionado para assegurar tanto a consistência das versões como da hierarquia de derivação pela verificação imediata das primitivas de atualização, poupando tempo e evitando que um grande número de operações precise ser desfeito.

Também foi detalhada a estratégia de gerenciamento do versionamento através de três níveis distintos: meta-esquema, intenção e extensão. Duas estratégias de consultas foram apresentadas para a recuperação dos dados da intenção e dos repositórios, respectivamente. A linguagem MSQL foi adotada para a recuperação das informações do metabanco, por ser uma linguagem de consulta desenvolvida especificamente para o versionamento de esquemas levando em consideração a solução múltiplos repositórios para o armazenamento. Já a TVQL foi utilizada para recuperar os dados da extensão, pois estes estão armazenados de acordo com as definições do TVM através dos controles dos metadados armazenados em cada repositório. No capítulo 8 é apresentado um estudo de caso desenvolvido sobre o protótipo implementado, descrito no capítulo 7.

7 TVMSE

Neste capítulo é apresentado o que foi implementado segundo as definições realizadas no capítulo 6, resultando no protótipo denominado *Temporal Versions Model Schema Evolution* (TVMSE). O TVMSE possui um painel de controle (figura 7.1) que permite a execução das diversas opções que compõem o sistema. A interface do TVMSE apresenta dois módulos principais que permitem gerenciar as versões de esquemas, e manipular os dados da intenção e da extensão respectivamente. Estes módulos serão apresentados detalhadamente a seguir.

FIGURA 7.1 - Painel de Controle TVMSE

A verificação das invariantes do sistema é gerenciada pela aplicação, sendo requisitada a cada operação do usuário, assegurando assim, uma evolução correta e consistente tanto dos esquemas como das instâncias. Este procedimento ocorre de forma transparente para ao usuário.

7.1 Módulo esquemas

O módulo esquemas contém as seguintes funcionalidades:

- especificação de um esquema;
- especificação de uma classe;
- especificação de um atributo;
- especificação de um relacionamento;
- correção e ou ativação de uma versão de esquema e seus componentes;
- derivação de uma versão de esquema;
- instanciação do repositório de uma versão de esquema.

7.1.1 Especificação de um esquema

A primeira opção disponível no TVMSE é a criação de um novo esquema, o que pode ser visualizado na figura 7.2. Nesta tela encontram-se os campos para o preenchimento do identificador da versão do esquema, do seu nome, da sua localização (se refere ao disco onde vai ser armazenado a base de dados relativa a esse esquema quando ele for ativado), do seu formato temporal e de seu intervalo de tempo de transação. Durante o preenchimento destes campos na tela, simultaneamente, vai se formando o *script* responsável para a criação da base desta versão de esquema.

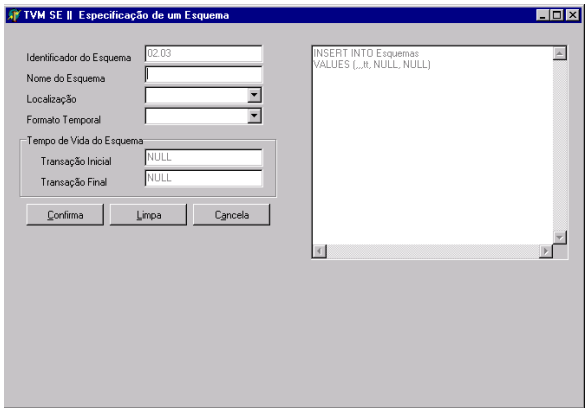


FIGURA 7.2 - Interface Especificação de um Esquema

A medida em que são preenchidas as especificações de uma nova versão de esquema, vão sendo inicializados os controles do meta-esquema de maneira transparente ao usuário.

7.1.2 Especificação de uma classe

Após definir as características gerais de um esquema pode-se definir as classes que o compõe por meio da interface de especificação de uma classe (figura 7.3). Nesta tela encontram-se os campos para o preenchimento do identificador da versão do esquema, do identificador da classe, do seu nome, de seu intervalo de tempo de transação, informações quanto ao tipo da classe, especificações para o uso de herança e, no caso de herança por extensão, a definição da correspondência entre as classes. Durante o preenchimento destes campos na tela, simultaneamente, vai sendo completado o *script* para a criação da base desta versão de esquema.

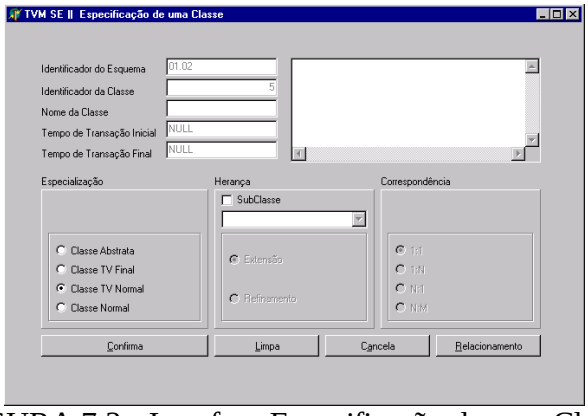


FIGURA 7.3 - Interface Especificação de uma Classe

Somente as classes definidas como TV Normal podem ter relacionamento de herança por extensão (neste caso é preciso definir a cardinalidade da correspondência entre as classes); as classes abstratas e normais (sem tempo e versão) só podem ter relacionamento de herança por refinamento. Uma classe TV Final não pode ser subclasse de outra.

As classes abstratas não são instanciadas nos repositórios dos esquemas. Suas especificações ficam armazenadas somente no meta-esquema.

7.1.3 Especificação de um atributo

Após definir as características gerais de uma classe, pode-se definir os seus atributos, por meio da interface de especificação de um atributo (figura 7.4). Nesta tela

encontram-se os campos para o preenchimento do identificador da versão do esquema, do identificador da classe, do identificador do atributo, do seu nome, tipo de dado, tamanho, se é temporal ou não, de seu intervalo de tempo de transação, e se é chave ou não. Durante o preenchimento destes campos na tela, simultaneamente, vai sendo completado o *script* para a criação da base desta versão de esquema.

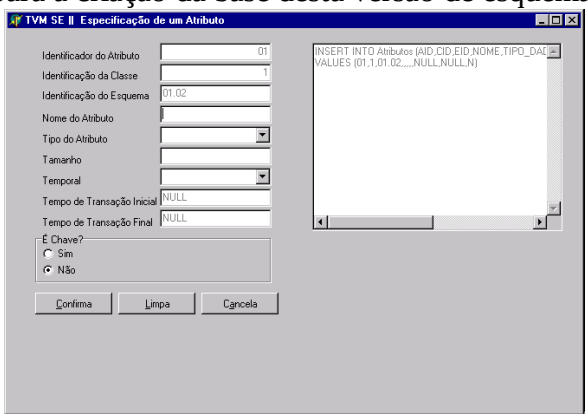


FIGURA 7.4 - Interface Especificação de um Atributo

Quando um atributo é definido como temporal, é definida a criação de uma tabela auxiliar no *script* para o armazenamento do seu histórico. Os atributos pertencentes a uma subclasse que possui relacionamento de herança por refinamento não podem possuir nomes iguais a nenhum atributo definido na superclasse.

7.1.4 Especificação de um relacionamento

Após definir as características gerais de um ou mais atributos, pode-se definir os relacionamentos por meio da interface de criação de um relacionamento (figura 7.5). Nesta tela encontram-se os campos para o preenchimento do identificador da versão do esquema, do identificador do relacionamento, do identificador da classe origem, do identificador do atributo de ligação da classe origem, do identificador da classe destino, do identificador do atributo de ligação da classe destino, do seu nome, de seu intervalo de tempo de transação. Durante o preenchimento destes campos na tela, simultaneamente, vai sendo completado o *script* para a criação da base desta versão de esquema.

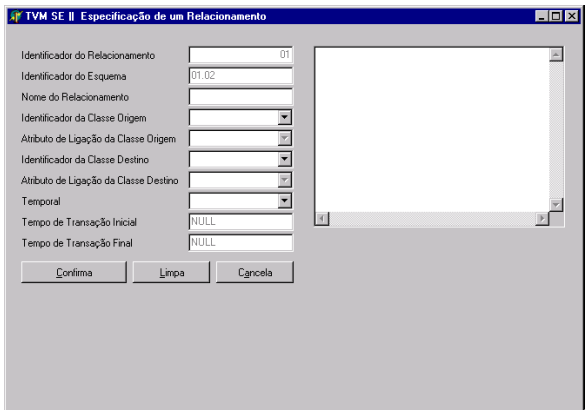


FIGURA 7.5 - Interface Especificação de um Relacionamento

7.1.5 Correção e/ou ativação de uma versão de esquema e de seus componentes

Após a definição de um esquema, é possível corrigi-lo (figura 7.6). Nesta tela pode-se selecionar o esquema para alteração ou para permitir a sua instanciação. As

operações de alteração nos componentes de uma versão de esquema podem ser de modificação de um componente existente, de inclusão de um novo ou de exclusão. A correção só permanece disponível para um esquema no estado `em trabalho`, isto é, uma versão de esquema que não sofreu derivação nem teve seu repositório criado na base.

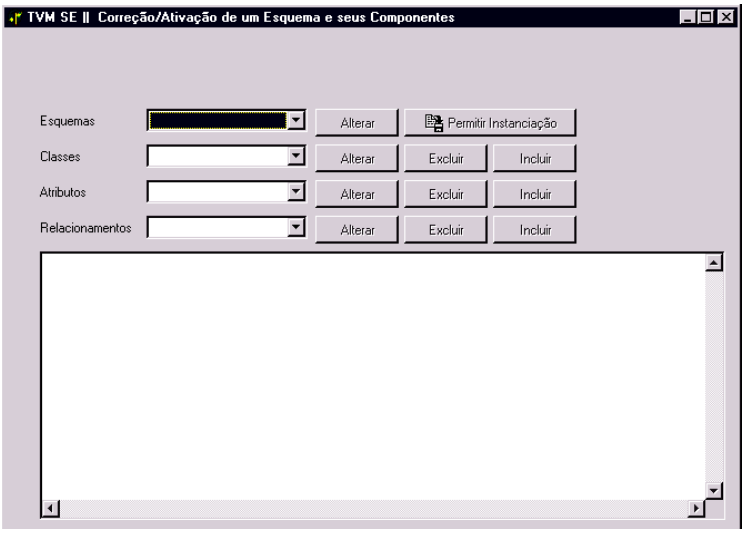


FIGURA 7.6 - Interface Correção/Ativação de um Esquema e seus Componentes

No momento em que o usuário permite a instanciação de uma versão de esquema, a aplicação gera um arquivo texto de nome "`script.txt`" para ser executado no DB2. Assim, é gerado o repositório com as especificações da versão de esquema escolhida.

7.1.6 Derivação de uma versão de esquema

Depois de realizadas as correções nas definições de uma versão de esquema, o usuário pode criar versões alternativas a partir dela. Esta operação denomina-se derivação (figura 7.7). No momento da derivação, a versão origem passa para o estado `congelada`, e a nova versão recebe o estado `em trabalho`, o que permite a sua correção.

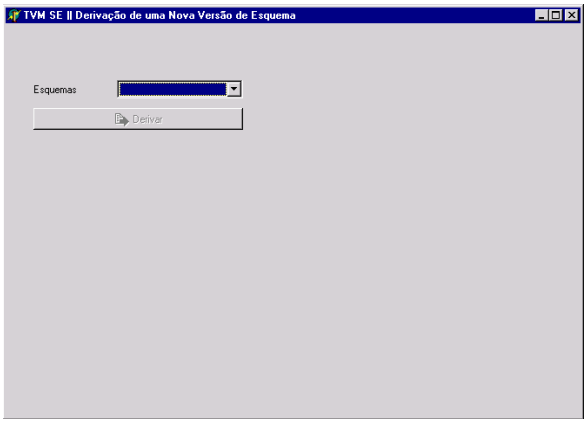


FIGURA 7.7 - Interface Derivação de uma Nova Versão de Esquema

Uma versão de esquema `congelada` tem seu tempo de transação final encerrado, o mesmo ocorrendo para todos os seus componentes. Se esta versão possuir um repositório associado, os objetos também terão encerrados os seus intervalos de transação e validade.

7.1.7 Instanciação do repositório de uma versão de esquema

No momento em que uma versão de esquema `estável` tem seu repositório de dados criado, este pode ser populado de acordo com as especificações do esquema que o gerou. A interface de instanciação (figura 7.8) apresenta tipos diferentes de tratamento para as classes normais e para as temporais versionadas, e também quanto à forma de preenchimento dos metadados presentes no repositório.

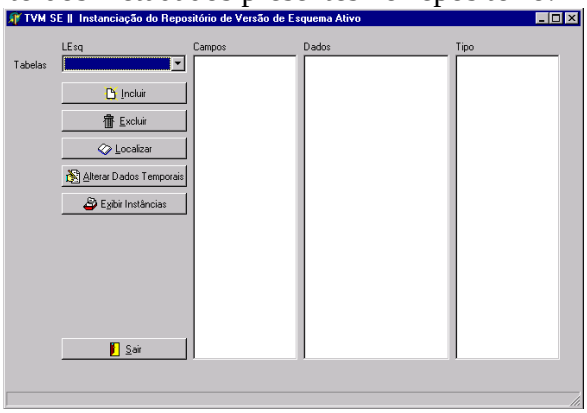


FIGURA 7.8 - Interface Instanciação do Repositório de uma Versão de Esquema Ativo

Na instanciação das classes normais, o `tvoid` gerado para a identificação dos objetos possui o número de versão igual a zero. Neste caso, as informações podem ser excluídas fisicamente e as alterações são sobrepostas às anteriores.

Na instanciação das classes temporais versionadas são observadas as seguintes características:

- o `tvoid` começa com o número de versão igual a um. No momento da alteração da informação são criados uma nova versão do objeto (o número da versão do `tvoid` é incrementado de um), o objeto versionado correspondente (número de versão igual a zero) e a tabela `VOC` que armazena as informações para o controle da evolução da base;
- o usuário pode definir uma versão como sendo a corrente (tabela `UserCurrentFlag`). Também é possível escolher uma versão como configuração de outra (tabela `TVConfig`);
- só é permitida a exclusão lógica, isto é, o estado da versão do objeto passa para `desativada`, seus tempos de validade e transação são encerrados, e o atributo `alive` recebe o valor não.

7.2 Módulo de consultas

O módulo de consultas contém as seguintes funcionalidades:

- definição de consulta à intenção;
- definição de consulta à extensão.

7.2.1 Consultas à intenção

Na interface para a especificação de uma consulta aos dados da intenção (figura 7.9) é possível recuperar, por intermédio do metabanco, informações de esquemas, classes, atributos e/ou campos. Para a consulta às informações do esquema versionado, a interface provém *radioboxes* para obter os resultados. Também é possível recuperar o histórico das informações definidas como temporais por meio da *checkbox* `Histórico`. A medida em que o usuário vai construindo a sua consulta, são apresentadas, simultaneamente, as respectivas consultas MSQL e SQL (que será

efetivamente executada no banco). O botão consultar, quando pressionado, apresenta o *grid* com os resultados desejados.

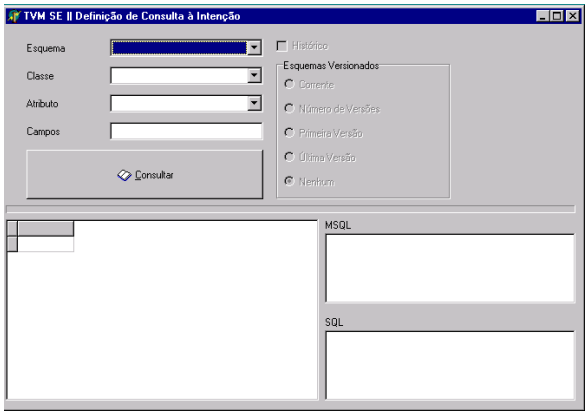


FIGURA 7.9 - Interface Definição de Consulta à Intenção

7.2.2 Consultas à extensão

Na interface para a especificação de uma consulta aos dados da extensão (figura 7.10) é possível escolher a(s) classe(s), o(s) atributo(s) a serem consultados no repositório do esquema escolhido. Para a consulta a classes temporais versionadas e seus atributos temporalizados, é possível requisitar seus históricos e versões. Também é permitido consultar as informações armazenadas nos metadados, como o conteúdo da tabela `VOC`, `AscDesc` e `PredSucc`. Neste tipo de consulta é possível estipular critérios e determinar os campos a serem retornados. A medida em que o usuário vai construindo a sua consulta, são apresentadas, simultaneamente, as respectivas consultas TVQL e SQL, esta adaptada para ser executada na base de dados. O botão consultar, quando pressionado, apresenta o *grid* com os resultados desejados.

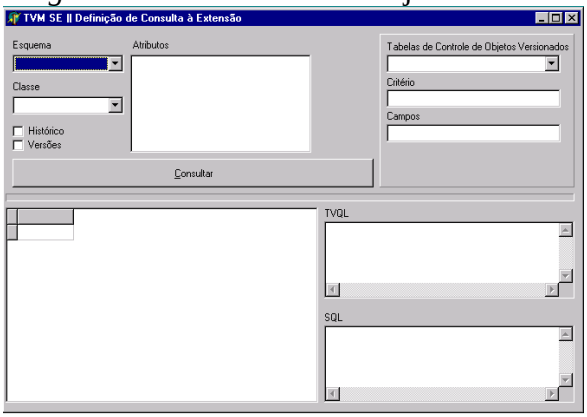


FIGURA 7.10 - Interface Definição de Consulta à Extensão

Cabe salientar que o protótipo implementado não cobre completamente as funcionalidades da TVQL, permitindo apenas consultas simples sobre dados históricos.

7.3 Considerações finais

Este capítulo apresentou as funcionalidades implementadas no protótipo TVMSE. Foram apresentados detalhadamente os módulos que o constituem. Neste contexto cabe lembrar que no módulo de consultas, mais especificamente em consultas à extensão, não é possível explorar todas as potencialidades definidas na TVQL. Isto ocorre devido à interface utilizada no TVMSE. Para a construção de consultas mais complexas tanto da TVQL como da MSQL seria necessário uma interface mais flexível onde o usuário fosse construindo suas consultas, e a aplicação, por sua vez, as fosse

interpretando e mapeando para a SQL correspondente. No capítulo 8 é apresentado um estudo de caso que demonstra o funcionamento do protótipo implementado.

8 Estudo de caso

O estudo de caso utilizado neste trabalho é a modelagem parcial de uma agência de publicidade. Nessa empresa são oferecidas campanhas de acordo com o perfil do cliente, sendo que várias estratégias de *marketing* são propostas até receber a aprovação do cliente.

8.1 Dados da intenção

O estudo de caso consiste no lançamento de um novo refrigerante para a temporada de verão, o *ShowCola*. O público alvo para este produto é composto de jovens e adolescentes. Com estes dados foi preparada uma estratégia de publicidade que consiste em uma campanha baseada no *site* da empresa e em *outdoors* próximos a *shoppings*, escolas, academias, etc. As características desta estratégia denota a primeira versão de esquema proposta, composta pelas classes `cliente`, `equipe`, `campanha`, `homepage` e `outdoor`.

Depois de uma reunião com a diretoria da *ShowCola* foi aprovada a proposta da agência, tendo sido incluído mais um meio de comunicação para o lançamento, que também deveria ser realizado através das rádios. Esta alteração gerou a segunda versão de esquema, composta pelas mesmas classes da primeira versão com acréscimo da classe `rádio`. O diagrama que representa a estratégia de *marketing* escolhida é apresentado na figura 8.1.

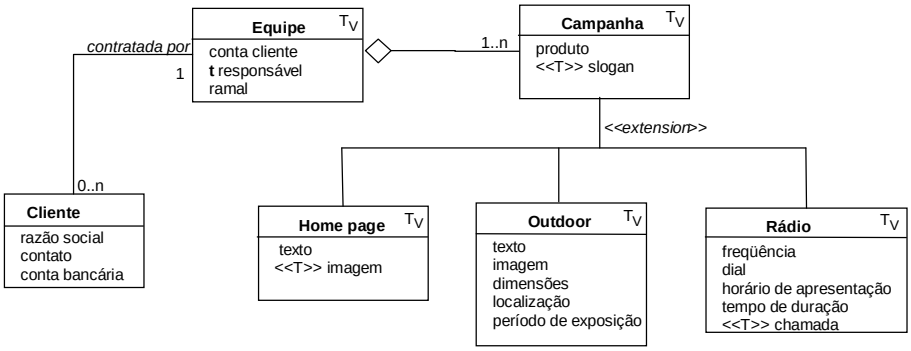


FIGURA 8.1 - Diagrama de classes do estudo de caso

A agência utiliza o TVMSE para modelar suas aplicações, sendo que para a campanha da *ShowCola* foram definidas duas estratégias diferentes. As tabelas 8.1, 8.2, 8.3 e 8.4 apresentam os dados armazenados no metabanco das tabelas `entidade`, `esquema` e do atributo temporal `estadoVersão` para cada uma das versões de esquema existentes.

TABELA 8.1 – Meta-esquema, tabela Entidade

ETID	EID
1	01.00
2	01.01

TABELA 8.2 - Meta-esquema, tabela Esquema

EID	NOME	LOCALIZAÇÃO	FORMATO_TEMPORAL	TTIMEI	TTIMEF
01.00	PROPOSTA1	C:	TEMPO DE TRANSAÇÃO	05/10/2002 09:00	15/11/2002 14:59
01.01	PROPOSTA2	C:	TEMPO DE TRANSAÇÃO	15/11/2002 15:00	NULL

TABELA 8.3 - Atributo temporal Estado, tabela Proposta1_EstadoVersão

EID	ESTADO	TTIMEI	TTIMEF
01.00	EM TRABALHO	10/09/2002 17:00	05/10/2002 08:59
01.00	ESTAVEL	05/10/2002 09:00	15/11/2002 14:59
01.00	CONGELADA	15/11/2002 15:00	NULL

TABELA 8.4 - Atributo temporal Estado, tabela Proposta2_EstadoVersão

EID	ESTADO	TTIMEI	TTIMEF
01.01	EM TRABALHO	05/10/2002 09:01	15/11/2002 14:59
01.01	ESTAVEL	15/11/2002 15:00	NULL

Cada versão de esquema possui a especificação das suas classes (tabela 8.5). As tabelas com as especificações dos atributos e relacionamentos não são apresentadas aqui por terem grande volume de dados repetidos, e por não serem utilizadas para representar as consultas do exemplo. Os *scripts* completos com as especificações das versões de esquema do estudo de caso estão nos anexos 3 e 4 respectivamente.

TABELA 8.5 - Meta-esquema, tabela Classe

CID	EID	NOME	TIPO	SUPER CLASSE	SUB CLASSE	TTIMEI	TTIMEF
1	01.00	CLIENTE	NORMAL	NÃO	NÃO	05/10/2002 09:00	15/11/2002 14:59
2	01.00	EQUIPE	TVNORMAL	NÃO	NÃO	05/10/2002 09:00	15/11/2002 14:59
3	01.00	CAMPANHA	TVNORMAL	SIM	NÃO	05/10/2002 09:00	15/11/2002 14:59
4	01.00	HOMEPAGE	TVNORMAL	NÃO	SIM	05/10/2002 09:00	15/11/2002 14:59
5	01.00	OUTDOOR	TVNORMAL	NÃO	SIM	05/10/2002 09:00	15/11/2002 14:59
1	01.01	CLIENTE	NORMAL	NÃO	NÃO	15/11/2002 15:00	NULL
2	01.01	EQUIPE	TVNORMAL	NÃO	NÃO	15/11/2002 15:00	NULL
3	01.01	CAMPANHA	TVNORMAL	SIM	NÃO	15/11/2002 15:00	NULL
4	01.01	HOMEPAGE	TVNORMAL	NÃO	SIM	15/11/2002 15:00	NULL
5	01.01	OUTDOOR	TVNORMAL	NÃO	SIM	15/11/2002 15:00	NULL
6	01.01	RADIO	TVNORMAL	NÃO	SIM	15/11/2002 15:00	NULL

Cada atributo temporal da tabela `EsquemaVersionado` é transformado em uma nova tabela que armazena os históricos da versão corrente (tabela 8.6), da primeira versão (tabela 8.7), da última versão (tabela 8.8), e do número de derivação (tabela 8.9).

TABELA 8.6 - Atributo temporal VersãoCorrente, tabela EsquemaVersionado

VSOID	EID	TTIMEI	TTIMEF
1	01.00	05/10/2002 09:00	15/11/2002 14:59
2	01.01	15/11/2002 15:00	NULL

TABELA 8.7 - Atributo temporal PrimeiraVersão, tabela EsquemaVersionado

VSOID	EID	TTIMEI	TTIMEF
1	01.00	05/10/2002 09:00	NULL

TABELA 8.8 - Atributo temporal UltimaVersão, tabela EsquemaVersionado

VSOID	EID	TTIMEI	TTIMEF
1	01.00	05/10/2002 09:00	15/11/2002 14:59
2	01.01	15/11/2002 15:00	NULL

TABELA 8.9 - Atributo temporal NroDerivação, tabela EsquemaVersionado

VSOID	EID	NRODERIVAÇÃO	TTIMEI	TTIMEF
1	01.00	0	05/10/2002 09:00	15/11/2002 14:59
1	01.00	1	15/11/2002 15:00	NULL
2	01.01	0	15/11/2002 15:00	NULL

8.2 Dados da extensão

Considerando que todas as tabelas já existem no banco de dados, e que os dados foram inseridos, essa seção apresenta os dados armazenados para a aplicação exemplo. O *script* de criação das tabelas está no Anexo 4.

Como primeira parte dos dados armazenados, são apresentadas as evoluções das instâncias de dois objetos versionados das classes `Campanha` e `Homepage` pertencentes à versão de esquema `Proposta2`.

Por determinação da agência, a campanha terá três *slogans* e imagens: um para o Natal, um para o *reveillon* e outro para o Carnaval.

Neste contexto, a tabela 8.10 apresenta a evolução das versões da instância do produto *ShowCola* dentro da tabela `campanha`.

TABELA 8.10 - Evolução do primeiro objeto campanha, tabela Campanha

TVOID	ALIVE	STATUS	CONFIGURATION	PRODUTO	SLOGAN	REFVOC
2,3,1	VERDADEIRO	ESTAVEL	FALSO	SHOWCOLA	O Natal vem vindo	2,3,0
2,3,2	VERDADEIRO	EM TRABALHO	FALSO	SHOWCOLA	2003 – sempre ShowCola	2,3,0
2,3,3	VERDADEIRO	EM TRABALHO	FALSO	SHOWCOLA	Gostoso é viver ShowCola	2,3,0

A tabela 8.11 apresenta a evolução das versões da instância do produto *ShowCola* dentro da tabela `homepage`.

TABELA 8.11 - Evolução do primeiro objeto homepage, tabela Homepage

TVOID	ALIVE	STATUS	CONFIGURATION	TEXTO	IMAGEM	REFVOC
2,4,1	VERDADEIRO	ESTAVEL	FALSO	Sempre ShowCola	Papainoel.gif	2,4,0
2,4,2	VERDADEIRO	EM TRABALHO	FALSO	Sempre ShowCola	Fogosartificio.gif	2,4,0
2,4,3	VERDADEIRO	EM TRABALHO	FALSO	Sempre ShowCola	Confeteserpentina.gif	2,4,0

A representação das tabelas 8.10 e 8.11 não é a mesma do repositório criado pois, além da tabela principal, existem as tabelas auxiliares para cada atributo temporal. Estas não foram apresentadas para evitar o excesso de informação.

Dados da aplicação

Esta seção apresenta os dados, para o produto *ShowCola*, das tabelas `homepage` e `campanha`, dos históricos temporais herdados:

- `alive` – instanciado quando o objeto é criado (tabela 8.12);
- `ascendant` e `descendant` – instanciado quando o objeto é criado (tabela 8.13);
- `status` – instanciado quando o objeto é criado e alterado quando novas versões são derivadas (tabela 8.14);
- `predecessor` e `successor` – instanciado quando o objeto é criado e alterado quando novas versões são derivadas (tabela 8.15).

TABELA 8.12 - Atributo temporal alive, tabelas Campanha e Homepage

Campanha

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,1	VERDADEIRO	15/11/2002 16:00		15/11/2002 16:00	
2,3,2	VERDADEIRO	16/11/2002 09:00		16/11/2002 09:00	
2,3,3	VERDADEIRO	17/11/2002 11:00		17/11/2002 11:00	

Homepage

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,4,1	VERDADEIRO	15/11/2002 16:10		15/11/2002 16:10	
2,4,2	VERDADEIRO	16/11/2002 09:10		16/11/2002 09:10	
2,4,3	VERDADEIRO	17/11/2002 11:10		17/11/2002 11:10	

TABELA 8.13 - Ascendentes e descendentes, tabela AscDesc

SEQAD	ASCENDANT	DECENDANT	TTIMEI	TTIMEF	VTIMEI	VTIMEF
1	NULL	2,3,1	15/11/2002 16:00		15/11/2002 16:00	
2	2,3,1	2,4,1	15/11/2002 16:10		15/11/2002 16:10	
3	NULL	2,3,2	16/11/2002 09:00		16/11/2002 09:00	
4	2,3,2	2,4,2	16/11/2002 09:10		16/11/2002 09:10	
5	NULL	2,3,3	17/11/2002 11:00		17/11/2002 11:00	
6	2,3,3	2,4,3	17/11/2002 11:10		17/11/2002 11:10	

TABELA 8.14 - Atributo temporal status, tabelas Campanha e Homepage

Campanha

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,1	EM TRABALHO	15/11/2002 16:00	16/11/2002 08:59	15/11/2002 16:00	
2,3,1	EM TRABALHO	15/11/2002 16:00		15/11/2002 16:00	16/11/2002 08:59
2,3,1	ESTAVEL	16/11/2002 09:00		16/11/2002 09:00	
2,3,2	EM TRABALHO	16/11/2002 09:00		16/11/2002 09:00	
2,3,3	EM TRABALHO	17/11/2002 11:00		17/11/2002 11:00	

Homepage

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,4,1	EM TRABALHO	15/11/2002 16:10	16/11/2002 09:09	15/11/2002 16:10	
2,4,1	EM TRABALHO	15/11/2002 16:10		15/11/2002 16:10	16/11/2002 09:09
2,4,1	ESTAVEL	16/11/2002 09:10		16/11/2002 09:10	
2,4,2	EM TRABALHO	16/11/2002 09:10		16/11/2002 09:10	
2,4,3	EM TRABALHO	17/11/2002 11:10		17/11/2002 11:10	

Obedecendo às Regras de Integridade Temporal definidas em (Moro, 2001), quando o objeto versionado é instanciado (número de versão igual a zero), o seu tempo de validade inicial é definido como igual ao tempo de validade da primeira versão do objeto (número de versão um).

A tabela 8.16 apresenta as instâncias dos controles dos objetos de `campanha` e `homepage` no instante atual. No momento da primeira derivação, é instanciado um objeto da classe `VersionedObjectControl` (abreviada `VOC`) que armazena as informações de controle dos objetos versionados. Seus valores são alterados nas derivações seguintes, na criação de configurações e na definição pelo usuário de uma versão como a corrente.

TABELA 8.15 - Predecessores e sucessores, tabela PredSuc

SEQPS	PREDECESSOR	SUCCESSOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
1	NULL	2,3,1	15/11/2002 16:00		15/11/2002 16:00	
2	2,3,1	NULL	15/11/2002 16:00	16/11/2002 08:59	15/11/2002 16:00	
3	2,3,1	NULL	15/11/2002 16:00		15/11/2002 16:00	16/11/2002 08:59
4	2,3,1	2,3,2	16/11/2002 09:00		16/11/2002 09:00	
5	2,3,2	NULL	16/11/2002 09:00		16/11/2002 09:00	
6	2,3,1	2,3,3	17/11/2002 11:00		17/11/2002 11:00	
7	2,3,3	NULL	17/11/2002 11:00		17/11/2002 11:00	
8	NULL	2,4,1	15/11/2002 16:10		15/11/2002 16:10	
9	2,4,1	NULL	15/11/2002 16:10	16/11/2002 09:09	15/11/2002 16:10	
10	2,4,1	NULL	15/11/2002 16:10		15/11/2002 16:10	16/11/2002 09:09
11	2,4,1	2,4,2	16/11/2002 09:10		16/11/2002 09:10	
12	2,4,2	NULL	16/11/2002 09:10		16/11/2002 09:10	
13	2,4,1	2,4,3	17/11/2002 11:10		17/11/2002 11:10	
14	2,4,3	NULL	17/11/2002 11:10		17/11/2002 11:10	

TABELA 8.16 - Controle dos objetos versionados, tabela VOC

TVOID	CONFIG COUNT	CURRENT VERSION	FIRST VERSION	LAST VERSION	NEXTV NUMBER	USER CURRENTF	VERSION COUNT
2,3,0	0	2,3,3	2,3,1	2,3,3	4	FALSO	3
2,4,0	0	2,4,3	2,4,1	2,4,3	4	FALSO	3

Os dados temporais destas instâncias são divididos em três grupos de tabelas:

- atributos `configurationCount` e `firstVersion` – são armazenados na instanciação e seus valores não são alterados no exemplo, conforme apresentado na tabela 8.17 e 8.18;

- `userCurrentFlag` – é armazenado na instanciação e seus valores são alterados quando o usuário estabelece uma versão como sendo a corrente, apresentado na tabela 8.19;
- atributos `currentVersion`, `firstVersion` e `versionCount` – são armazenados na instanciação e alterados nas derivações, conforme apresentado nas tabelas 8.20, 8.21, 8.22.

TABELA 8.17 - Controle dos objetos versionados, tabela VOCconfigCount

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	0	16/11/2002 09:00		16/11/2002 09:00	
2,4,0	0	16/11/2002 09:10		16/11/2002 09:10	

TABELA 8.18 - Controle dos objetos versionados, tabela VOCfirstVersion

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	2,3,1	16/11/2002 09:00		16/11/2002 09:00	
2,4,0	2,4,1	16/11/2002 09:10		16/11/2002 09:10	

TABELA 8.19 - Controle dos objetos versionados, tabela VOCuserCurrentF

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	FALSO	16/11/2002 09:00		16/11/2002 09:00	
2,4,0	FALSO	16/11/2002 09:10		16/11/2002 09:10	

TABELA 8.20 - Controle dos objetos versionados, tabela VOCcurrentVersion

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	2,3,2	16/11/2002 09:00	17/11/2002 10:59	16/11/2002 09:00	
2,3,0	2,3,2	16/11/2002 09:00		16/11/2002 09:00	17/11/2002 10:59
2,3,0	2,3,3	17/11/2002 11:00		17/11/2002 11:00	
2,4,0	2,4,2	16/11/2002 09:10	17/11/2002 11:09	16/11/2002 09:10	
2,4,0	2,4,2	16/11/2002 09:10		16/11/2002 09:10	17/11/2002 11:09
2,4,0	2,4,3	17/11/2002 11:10		17/11/2002 11:10	

TABELA 8.21 - Controle dos objetos versionados, tabela VOClastVersion

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	2,3,2	16/11/2002 09:00	17/11/2002 10:59	16/11/2002 09:00	
2,3,0	2,3,2	16/11/2002 09:00		16/11/2002 09:00	17/11/2002 10:59
2,3,0	2,3,3	17/11/2002 11:00		17/11/2002 11:00	
2,4,0	2,4,2	16/11/2002 09:10	17/11/2002 11:09	16/11/2002 09:10	
2,4,0	2,4,2	16/11/2002 09:10		16/11/2002 09:10	17/11/2002 11:09
2,4,0	2,4,3	17/11/2002 11:10		17/11/2002 11:10	

TABELA 8.22 - Controle dos objetos versionados, tabela VOCversionCount

TVOID	VALOR	TTIMEI	TTIMEF	VTIMEI	VTIMEF
2,3,0	2	16/11/2002 09:00	17/11/2002 10:59	16/11/2002 09:00	
2,3,0	2	16/11/2002 09:00		16/11/2002 09:00	17/11/2002 10:59
2,3,0	3	17/11/2002 11:00		17/11/2002 11:00	
2,4,0	2	16/11/2002 09:10	17/11/2002 11:09	16/11/2002 09:10	
2,4,0	2	16/11/2002 09:10		16/11/2002 09:10	17/11/2002 11:09
2,4,0	3	17/11/2002 11:10		17/11/2002 11:10	

8.3 Consultas

Nesta seção são apresentadas algumas possíveis consultas sobre os dados armazenados e o resultado obtido pelo TVMSE. Por omissão, a data de realização da consulta é a data de hoje (março de 2003) e o estado da base de dados é o atual (tempo de transação final em aberto).

8.3.1 Exemplos de consulta à intenção (MSQL)

1) Retornar os nomes das classes que compõem a versão de esquema 01.01:

```
SELECT [METABANCO[5] : NOME]
FROM CLASSE
WHERE CLASSE.EID = '01.01';
```

NOME
CLIENTE
EQUIPE
CAMPANHA
HOMEPAGE
OUTDOOR

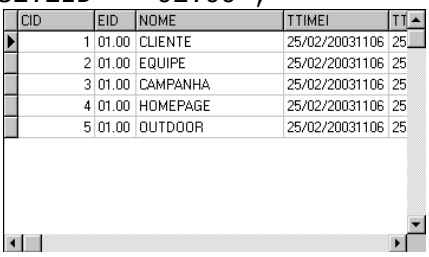
FIGURA 8.2 - Resultado obtido na consulta 1 (intenção)

2) Retornar as especificações das classes que compõem a versão de esquema 01.00:

```
SELECT *
```

[5] Metabanco é o nome do repositório da base de dados que armazena as informações pertencentes ao meta-esquema.

```
FROM [METABANCO : CLASSE]
WHERE CLASSE.EID = '01.00';
```

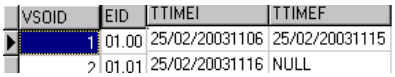


CID	EID	NOME	TTIMEI	TT
1	01.00	CLIENTE	25/02/20031106	25
2	01.00	EQUIPE	25/02/20031106	25
3	01.00	CAMPANHA	25/02/20031106	25
4	01.00	HOMEPAGE	25/02/20031106	25
5	01.00	OUTDOOR	25/02/20031106	25

FIGURA 8.3 - Resultado obtido na consulta 2 (intenção)

3) Retornar o histórico das versões correntes de esquema:

```
SELECT EVER [METABANCO : VERSAOCORRENTE]
FROM ESQUEMAVERSIONADO_VERSAOCORRENTE AS E
```

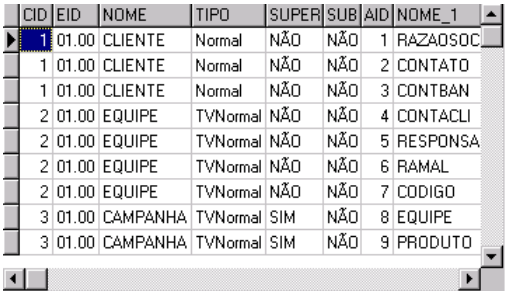


VSOID	EID	TTIMEI	TTIMEF
1	01.00	25/02/20031106	25/02/20031115
2	01.01	25/02/20031116	NULL

FIGURA 8.4 - Resultado obtido na consulta 3 (intenção)

4) Retornar as especificações das classes e de seus atributos na versão de esquema 01.00:

```
SELECT *
FROM [METABANCO : CLASSE] AS C, [METABANCO : ATRIBUTO] AS A
WHERE C.EID='01.00' AND C.EID=A.EID AND C.CID= A.CID;
```

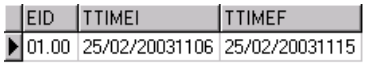


CID	EID	NOME	TIPO	SUPER	SUB	AID	NOME_1
1	01.00	CLIENTE	Normal	NÃO	NÃO	1	RAZAOSOC
1	01.00	CLIENTE	Normal	NÃO	NÃO	2	CONTATO
1	01.00	CLIENTE	Normal	NÃO	NÃO	3	CONTBAN
2	01.00	EQUIPE	TVNormal	NÃO	NÃO	4	CONTACLI
2	01.00	EQUIPE	TVNormal	NÃO	NÃO	5	RESPONSA
2	01.00	EQUIPE	TVNormal	NÃO	NÃO	6	RAMAL
2	01.00	EQUIPE	TVNormal	NÃO	NÃO	7	CODIGO
3	01.00	CAMPANHA	TVNormal	SIM	NÃO	8	EQUIPE
3	01.00	CAMPANHA	TVNormal	SIM	NÃO	9	PRODUTO

FIGURA 8.5 - Resultado obtido na consulta 4 (intenção)

5) Retornar o período de vida da versão de esquema 01.00:

```
SELECT [METABANCO : TTIMEI, METABANCO : TTIMEF]
FROM ESQUEMA
WHERE ESQUEMA.EID ='01.00'
```

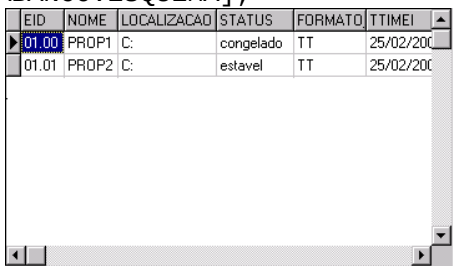


EID	TTIMEI	TTIMEF
01.00	25/02/20031106	25/02/20031115

FIGURA 8.6 - Resultado obtido na consulta 5 (intenção)

6) Retornar as versões de esquema definidas na aplicação:

```
SELECT *
FROM [METABANCO:ESQUEMA];
```



EID	NOME	LOCALIZACAO	STATUS	FORMATO	TTIMEI
01.00	PROP1	C:	congelado	TT	25/02/200
01.01	PROP2	C:	estavel	TT	25/02/200

FIGURA 8.7 - Resultado obtido na consulta 6 (intenção)

8.3.2 Exemplos de consulta à extensão (TVQL)

1) Retornar o identificador e a imagem atual da tabela `homepage`:

```
SET SCHEMA PROPOSTA2
SELECT IMAGEM
FROM HOMEPAGE;
```

TVOID	IMAGEM
2,4,3	confeteserpentina.gif

FIGURA 8.8 - Resultado obtido na consulta 1 (extensão)

2) Retornar todas as informações dos objetos que possuem a versão ‘2,3,1’ como predecessora:

```
SET SCHEMA PROPOSTA2
SELECT *
FROM PREDSUCC
WHERE PREDSUCC.PREDECESSOR = ‘2,3,1’;
```

SEQPS	PREDECESSOR	SUCESSOR	TTIMEI	TTIMEF
2	2,3,1	NULL	28/02/20030909	01/03/2
3	2,3,1	NULL	28/02/20030909	NULL
4	2,3,1	2,3,2	01/03/20031010	NULL
6	2,3,1	2,3,3	02/03/20031415	NULL

FIGURA 8.9 - Resultado obtido na consulta 2 (extensão)

3) Retornar os identificadores e os slogans das versões que estão no estado em `trabalho` da tabela `campanha`:

```
SET SCHEMA PROPOSTA2
SELECT V.TVOID, V.SLOGAN
FROM CAMPANHA.VERSIONS V
WHERE V.ISWORKING;
```

TVOID	SLOGAN
2,3,2	2003 - Sempre Showcola
2,3,3	Gostoso é viver Showcola

FIGURA 8.10 - Resultado obtido na consulta 3 (extensão)

8.4 Considerações finais

Neste capítulo foi apresentado um estudo de caso que permitiu a visualização do gerenciamento realizado no TVMSE. Mediante a modelagem das propostas de *marketing* de uma agência de publicidade foi possível observar a evolução dos esquemas e de suas instâncias. Também foram apresentados alguns exemplos de consulta suportados pela aplicação.

9 Conclusões

A maioria das aplicações está em constante alteração exigindo que os seus esquemas conceituais também sejam modificados dinamicamente. A manutenção coerente dos esquemas e de seus dados é uma tarefa bastante complexa, ainda mais quando se considera o versionamento de esquemas. Este tipo de controle envolve muitos detalhes para ser efetuado, pois um esquema pode ser alterado em diversos aspectos e todos estes devem ser gerenciados, ou seja, a aplicação deverá garantir uma visão íntegra e consistente dos dados conforme o esquema determinado. Ainda considerando versionamento, o trabalho utiliza o TVM como modelo de dados, o que define versionamento tanto na intenção quanto na extensão.

Neste contexto, o presente trabalho apresentou as seguintes contribuições:

- a identificação de quatro alternativas distintas de derivação de versões quando se trata o versionamento de esquemas;
- a proposta de uma estratégia de implementação que efetue o gerenciamento do versionamento em três níveis – meta-esquema, intenção e extensão;
- a implementação de um protótipo que permite a evolução de esquemas de acordo com a alternativa de derivação escolhida, onde as versões de esquema são de tempo de transação e os dados são TVM. Foi utilizada a estratégia de múltiplos repositórios para o armazenamento das versões de esquemas e de suas instâncias, e o gerenciamento entre estas é síncrono;
- a proposta de duas estratégias para consulta aos dados. Na intenção é utilizada a MSQL e para a recuperação das instâncias nos seus respectivos repositórios é utilizada a TVQL;
- a inclusão da cláusula `ever` na MSQL para que esta possa realizar consultas históricas através dos diferentes repositórios das versões de esquema;
- uma extensão da TVQL para selecionar o repositório da versão de esquema utilizada para consultar as versões de instâncias;
- a implementação do gerenciamento da evolução tanto dos esquemas como das instâncias via aplicação, de modo que, esta pode ser utilizada em outros bancos de dados, não somente com o DB2.

9.1 Trabalhos Futuros

Observando o trabalho realizado é possível identificar aspectos que podem ser aperfeiçoados, tais como:

- a implementação no mesmo gerenciador da possibilidade de escolha de uma das alternativas de derivação, e da solução de armazenamento;
- a possibilidade do gerenciamento assíncrono;
- a extensão da linguagem de especificação de classes do TVM para especificação de esquemas;
- a implementação de um módulo de consultas TVQL e MSQL onde haja a possibilidade de consultar a base utilizando todos os recursos destas linguagens;
- a extensão da TVQL para possibilitar consultas entre diferentes versões de esquema, no nível do meta-esquema;
- a implementação da aplicação em um banco de dados orientado a objetos.

Anexo 1 Script do Metabanco

```
CREATE DATABASE METABD
CONNECT TO METABD

CREATE TABLE ESQUEMAS
(EID              VARCHAR(5) NOT NULL,
 NOME             VARCHAR(20),
 LOCALIZACAO      VARCHAR(50),
 FORMATO_TEMPORAL VARCHAR(4),
 STATUS           VARCHAR (20),
 VSOID            INTEGER,
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT ESQUEMAPK PRIMARY KEY (EID, TTIMEI));

CREATE TABLE ENTIDADES
(ETID             INTEGER,
 EID              VARCHAR(5) NOT NULL,
 NOME             VARCHAR(20),
 CONSTRAINT ENTIDADESPK PRIMARY KEY (ETID),
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE CLASSES
(CID              INTEGER NOT NULL,
 EID              VARCHAR(5),
 NOME             VARCHAR(20),
 TIPO             VARCHAR(20),
 SUPERCLASSE      VARCHAR(20),
 SUBCLASSE        VARCHAR(20),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT CLASSESPK PRIMARY KEY (CID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE ATRIBUTOS
(AID              INTEGER NOT NULL,
 CID              INTEGER,
 EID              VARCHAR(5),
 NOME             VARCHAR(20),
 TIPO_DADO        VARCHAR(20),
 TEMPORAL         VARCHAR(1),
 TAMANHO          VARCHAR(5),
 CHAVE            VARCHAR(1),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT ATRIBUTOSPK PRIMARY KEY (AID, TTIMEI),
 FOREIGN KEY CID REFERENCES CLASSES,
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE RELACIONAMENTOS
(RID              INTEGER NOT NULL,
 EID              VARCHAR(5),
 NOME             VARCHAR(20),
 C1ID             INTEGER,
 A1ID             INTEGER,
 C2ID             INTEGER,
```

```
 A2ID             INTEGER,
 TIPO             VARCHAR(12),
 TEMPORAL         VARCHAR(1),
 CORRESPONDENCIA  VARCHAR(3),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT RELACIONAMENTOSPK PRIMARY KEY (RID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA,
 FOREIGN KEY C1ID, C2ID REFERENCES CLASSES,
 FOREIGN KEY A1ID, A2ID REFERENCES ATRIBUTOS);


CREATE TABLE EVCORRENTE
(VSOID            INTEGER NOT NULL,
 EID              VARCHAR(5),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT EVCORRENTEPK PRIMARY KEY (VSOID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE EVNUMVER
(VSOID            INTEGER NOT NULL,
 EID              VARCHAR(5),
 NUMVER           INTEGER,
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT EVNUMVERPK PRIMARY KEY (VSOID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE EVPRIM
(VSOID            INTEGER NOT NULL,
 EID              VARCHAR(5),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT EVPRIMPK PRIMARY KEY (VSOID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA);

CREATE TABLE EVULT
(VSOID            INTEGER NOT NULL,
 EID              VARCHAR(5),
 TTIMEI           VARCHAR(14) NOT NULL,
 TTIMEF           VARCHAR(14),
 CONSTRAINT EVULTPK PRIMARY KEY (VSOID, TTIMEI),
 FOREIGN KEY EID REFERENCES ESQUEMA);
```

Anexo 2 Script dos Metadados

```
CREATE TABLE VOC
(TVOID            Varchar(20) NOT NULL,
 NEXTNUMBER       Varchar(20),
 CONSTRAINT VOCPK PRIMARY KEY (TVOID));

CREATE TABLE VOCVERCORR
(TVOID            Varchar(20) NOT NULL,
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCVERCORRPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMCONFIG
(TVOID            Varchar(20) NOT NULL,
 NUMCONFIG        Varchar(20),
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCNUMCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMDERIV
(TVOID            Varchar(20) NOT NULL,
 NUMDERIV         Integer,
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCNUMDERIVPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCPRIMVER
(TVOID            Varchar(20) NOT NULL,
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCPRIMVERPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCULTVER
(TVOID            Varchar(20) NOT NULL,
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCULTVERPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC)

CREATE TABLE VOCUSERCURRF
(TVOID            Varchar(20) NOT NULL,
 TTIMEI           Varchar(14) NOT NULL,
```

```
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT VOCUSERCURRFPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE PREDSUCC
(SEQPS            Integer NOT NULL,
 PREDECESSOR      Varchar(20),
 SUCESSOR         Varchar(20),
 TTIMEI           Varchar(14),
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14),
 VTIMEF           Varchar(14),
 CONSTRAINT PREDSUCCPK PRIMARY KEY (SEQPS));

CREATE TABLE ASCDESC
(SEQAD            Integer NOT NULL,
 ASCENDENTE       Varchar(20),
 DESCENDENTE      Varchar(20),
 TTIMEI           Varchar(14),
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14),
 VTIMEF           Varchar(14),
 CONSTRAINT ASCDESCPK PRIMARY KEY (ASCDESC));

CREATE TABLE CLATVALIVE
(TVOID            Varchar(20) NOT NULL,
 ALIVE            Varchar(1),
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT CLATVALIVEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLATVCONFIG
(TVOID            Varchar(20) NOT NULL,
 CONFIGURACAO     Varchar(1),
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT CLATVCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLATVSTATUS
(TVOID            Varchar(20) NOT NULL,
 STATUS           Varchar(20),
 TTIMEI           Varchar(14) NOT NULL,
 TTIMEF           Varchar(14),
 VTIMEI           Varchar(14) NOT NULL,
 VTIMEF           Varchar(14),
 CONSTRAINT CLATVSTATUSPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
 FOREIGN KEY TVOID REFERENCES VOC);
```

Anexo 3 Script do Estudo de Caso (versão de esquema 01.00)

```
CREATE DATABASE PROP1 ON C:
CONNECT TO PROP1

CREATE TABLE CLIENTE
 (ID              Integer NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  RAZAOSOC        Varchar(50),
  CONTATO         Varchar(50),
  CONTBAN         Varchar(30) NOT NULL,
  CONSTRAINT CLIENTEPK PRIMARY KEY (ID, TTIMEI, CONTBAN);


CREATE TABLE EQUIPE
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONTACLI        Varchar(30),
  RAMAL           Varchar(8),
  CODIGO          Integer NOT NULL,
  CONSTRAINT EQUIPEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI, CODIGO),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CAMPANHA
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  EQUIPE          Integer,
  PRODUTO         Varchar(50),
  CONSTRAINT CAMPANHAPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE HOMEPAGE
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  TEXTO           Varchar(50),
  CONSTRAINT HOMEPAGEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE OUTDOOR
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  TEXTO           Varchar(50),
  IMAGEM          Varchar(50),
  DIMENSOES       Varchar(30),
```

```
  LOCAL           Varchar(50),
  PERIODO         Varchar(20),
  CONSTRAINT OUTDOORPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE EQUIPE_RESPONSA
 (TVOID           Varchar(20) NOT NULL,
  RESPONSA        Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT EQUIPE_RESPONSAPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES EQUIPE);

CREATE TABLE CAMPANHA_SLOGAN
 (TVOID           Varchar(20) NOT NULL,
  SLOGAN          Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT CAMPANHA_SLOGANPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES CAMPANHA);

CREATE TABLE HOMEPAGE_IMAGEM
 (TVOID           Varchar(20) NOT NULL,
  IMAGEM          Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT HOMAPEGE_IMAGEMPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES HOMEPAGE);

CREATE TABLE VOC
(TVOID            Varchar(20) NOT NULL,
 NEXTNUMBER       Varchar(20),
 CONSTRAINT VOKPK PRIMARY KEY (TVOID));

CREATE TABLE VOCVERCORR
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCVERCORRPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMCONFIG
 (TVOID           Varchar(20) NOT NULL,
  NUMCONFIG       Varchar(20),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCNUMCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMDERIV
 (TVOID           Varchar(20) NOT NULL,
  NUMDERIV        Integer,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
```

```
  CONSTRAINT VOCNUMDERIVPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENECES VOC);

CREATE TABLE VOCPRIMVER
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCPRIMVERPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCUSERCURRF
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCUSERCURRF PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCULTVER
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCULTVERPK PRIMARY KEY (TVOID,TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE PREDSUCC
 (SEQPS           Integer NOT NULL,
  PREDECESSOR     Varchar(20),
  SUCESSOR        Varchar(20),
  TTIMEI          Varchar(14),
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14),
  VTIMEF          Varchar(14),
  CONSTRAINT PREDSUCCPK PRIMARY KEY (SEQPS));

CREATE TABLE ASCDESC
 (SEQAD           Integer NOT NULL,
  ASCENDENTE      Varchar(20),
  DESCENDENTE     Varchar(20),
  TTIMEI          Varchar(14),
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14),
  VTIMEF          Varchar(14),
  CONSTRAINT ASCDESCPK PRIMARY KEY (SEQAD));

CREATE TABLE CLATVALIVE
 (TVOID           Varchar(20) NOT NULL,
  ALIVE           Varchar(1),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVALIVEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);
```

```
CREATE TABLE CLATVCONFIG
 (TVOID           Varchar(20) NOT NULL,
  CONFIGURACAO    Varchar(1),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLATVSTATUS
 (TVOID           Varchar(20) NOT NULL,
  STATUS          Varchar(20),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVSTATUSPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLIENTE_EQUIPE
 (ID              Integer NOT NULL,
  ID1             Varchar(20),
  ID2             Varchar(20),
  CONSTRAINT CLIENTE_EQUIPEPK PRIMARY KEY (ID),
  FOREIGN KEY ID1 REFERENCES CLIENTE,
  FOREIGN KEY ID2 REFERENCES EQUIPE);

CREATE TABLE EQUIPE_CAMPANHA
 (ID              Integer NOT NULL,
  ID1             Varchar(20),
  ID2             Varchar(20),
  CONSTRAINT EQUIPE_CAMPANHA PRIMARY KEY (ID),
  FOREIGN KEY ID1 REFERENCES EQUIPE,
  FOREIGN KEY ID2 REFERENCES CAMPANHA);
```

Anexo 4 Script do Estudo de Caso (versão de esquema 01.01)

```
CREATE DATABASE PROP2 ON C:
CONNECT TO PROP2

CREATE TABLE CLIENTE
 (ID              Integer NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  RAZAOSOC        Varchar(50),
  CONTATO         Varchar(50),
  CONTBAN         Varchar(30) NOT NULL,
  CONSTRAINT CLIENTEPK PRIMARY KEY (ID, TTIMEI, CONTBAN));

CREATE TABLE EQUIPE
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONTACLI        Varchar(30),
  RAMAL           Varchar(8),
  CODIGO          Integer NOT NULL,
  CONSTRAINT EQUIPEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI, CODIGO),
  FOREIGN KEY TVOID REFERENCES EQUIPE);

CREATE TABLE CAMPANHA
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  EQUIPE          Integer,
  PRODUTO         Varchar(50),
  CONSTRAINT CAMPANHAPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE HOMEPAGE
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  TEXTO           Varchar(50),
  CONSTRAINT HOMEPAGEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE OUTDOOR
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  TEXTO           Varchar(50),
  IMAGEM          Varchar(50),
  DIMENSOES       Varchar(30),
  LOCAL           Varchar(50),
```

```
  PERIODO         Varchar(20),
  CONSTRAINT OUTDOORPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE RADIO
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  FREQ            Decimal(6,2),
  DIAL            Varchar(3),
  HORARIO         Varchar(20),
  DURACAO         Varchar(20),
  CONSTRAINT RADIOPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE EQUIPE_RESPONSA
 (TVOID           Varchar(20) NOT NULL,
  RESPONSA        Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT EQUIPE_RESPONSAPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES EQUIPE);

CREATE TABLE CAMPANHA_SLOGAN
 (TVOID           Varchar(20) NOT NULL,
  SLOGAN          Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT CAMPANHA_SLOGANPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES CAMPANHA);

CREATE TABLE HOMEPAGE_IMAGEM
 (TVOID           Varchar(20) NOT NULL,
  IMAGEM          Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT HOMEPAGE_IMAGEMPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES HOMEPAGE);

CREATE TABLE RADIO_CHAMADA
 (TVOID           Varchar(20) NOT NULL,
  CHAMADA         Varchar(50),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  CONSTRAINT RADIO_CHAMADAPK PRIMARY KEY (TVOID, TTIMEI),
  FOREIGN KEY TVOID REFERENCES RADIO);

CREATE TABLE VOC
 (TVOID           Varchar(20) NOT NULL,
  NEXTNUMBER      Varchar(20),
  CONSTRAINT VOCPK PRIMARY KEY (TVOID));

CREATE TABLE VOCVERCORR
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
```

```
  CONSTRAINT VOCVERCORRPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMCONFIG
 (TVOID           Varchar(20) NOT NULL,
  NUMCONFIG       Varchar(20),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCNUMCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCNUMDERIV
 (TVOID           Varchar(20) NOT NULL,
  NUMDERIV        Integer,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCNUMDERIVPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCPRIMVER
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCPRIMVERPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCUSERCURRF
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCUSERCURRFPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE VOCULTVER
 (TVOID           Varchar(20) NOT NULL,
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT VOCULTVERPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE PREDSUCC
 (SEQPS           Integer NOT NULL,
  PREDECESSOR     Varchar(20),
  SUCESSOR        Varchar(20),
  TTIMEI          Varchar(14),
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14),
  VTIMEF          Varchar(14),
  CONSTRAINT PREDSUCCPK PRIMARY KEY (SEQPS));
```

```
CREATE TABLE ASCDESC
 (SEQAD           Integer NOT NULL,
  ASCENDENTE      Varchar(20),
  DESCENDENTE     Varchar(20),
  TTIMEI          Varchar(14),
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14),
  VTIMEF          Varchar(14),
  CONSTRAINT ASCDESCPK PRIMARY KEY (ASCDESC));

CREATE TABLE CLATVALIVE
 (TVOID           Varchar(20) NOT NULL,
  ALIVE           Varchar(1),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVALIVEPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLATVCONFIG
 (TVOID           Varchar(20) NOT NULL,
  CONFIGURACAO    Varchar(1),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVCONFIGPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLATVSTATUS
 (TVOID           Varchar(20) NOT NULL,
  STATUS          Varchar(20),
  TTIMEI          Varchar(14) NOT NULL,
  TTIMEF          Varchar(14),
  VTIMEI          Varchar(14) NOT NULL,
  VTIMEF          Varchar(14),
  CONSTRAINT CLATVSTATUSPK PRIMARY KEY (TVOID, TTIMEI, VTIMEI),
  FOREIGN KEY TVOID REFERENCES VOC);

CREATE TABLE CLIENTE_EQUIPE
 (ID              Integer NOT NULL,
  ID1             Varchar(20),
  ID2             Varchar(20),
  CONSTRAINT CLIENTE_EQUIPEPK PRIMARY KEY (ID),
  FOREIGN KEY ID1 REFERENCES CLIENTE,
  FOREIGN KEY ID2 REFERENCES EQUIPE);
CREATE TABLE EQUIPE_CAMPANHA
 (ID              Integer NOT NULL,
  ID1             Varchar(20),
  ID2             Varchar(20),
  CONSTRAINT EQUIPE_CAMPANHAPK PRIMARY KEY (ID),
  FOREIGN KEY ID1 REFERENCES EQUIPE,
  FOREIGN KEY ID2 REFERENCES CAMPANHA);
```

Referências

ANGONESE, S. F. **Gerenciamento Temporal de Versões de Esquemas**. 2000. 92p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

BÖHLEN, M.H.; JENSEN, C.S.; SNODGRASS, R.T. Temporal Statement Modifiers. **ACM Transactions on Database Systems**, New York, v. 25, n. 4, p. 407 – 456, Dec. 2000.

CAMOLESI JÚNIOR, L.; TRAINA JÚNIOR, C. Evolução de Esquemas de Dados – Um panorama Amplo de Aspectos Técnicos e Gerenciais. In: SBBD, 1996, São Carlos, SP. **Anais...** São Carlos: ICMS/USP, 1996. p. 1-19.

CHOMICKI, J. **Temporal Query Languages:** a survey. Manhattan: Kansas State University, 1995. 38 p. Disponível em: <citeseer.nj.nec.com/article/chomicki95temporal.html>. Acesso em: 10 set. 2002.

CHOMICKI, J.; TOMAN, D.; BÖHLEN, M.H. Querying ARSQL Databases with Temporal Logic. **ACM Transactions on Database Systems**, New York, v. 26, n. 2, p.145 – 178, June 2001.

DE CASTRO, C.; GRANDI, F.; SCALAS, M.R. On Schema Versioning in Temporal Databases**.** In: INTERNATIONAL WORKSHOP ON TEMPORAL DATABASES, 1995, Zurich. **Recent Advances in Temporal Databases**: proceedings. Berlin: Springer-Verlag, 1995. p. 272-291.

DE CASTRO, C.; GRANDI, F.; SCALAS, M.R. Schema Versioning for Multitemporal Relational Databases. **Information Systems**, Oxford, v. 22, n. 5, p. 249-290, July 1997.

DE CASTRO, C. A First Approach to Temporal Predicate Locking for Concurrency Detection in Temporal Relational Databases supporting Schema Versioning. In: SEMINAR ON CURRENT TRENDS IN THEORY AND PRACTICE OF INFORMATICS, SOFSEM, 24., 1997, Milovy. **Theory and Practice of Informatics**: proceedings. Berlin: Springer-Verlag, 1997.

EDELWEISS, N; CASTILHO, J.M.V; OLIVEIRA, J.P.M. Temporal Aspects of Conceptual Schema Evolution. In: CONFERENCIA INTERNACIONAL DE LA SOCIEDAD CHILENA DE CIENCIA DE LA COMPUTACION, 15., 1995. **Actas**. Santiago: Sociedad Chilena de Ciencia de la Computacion, 1995. p. 187-197.

EDELWEISS, N; OLIVEIRA, J.P.M. **Modelagem de aspectos temporais de sistemas de informação.** Recife: UFPE-DI, 1994. 163p.

ELMASRI, R; NAVATHE, S.B. **Fundamentals of Database Systems**. 3rd ed. Redwood City: Addison-Wesley Longman, 2000. 955p.

GALANTE, R. M. **Um modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões**. 1998. 96p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

GALANTE, R.M.; ROMA, A.B.S.; JANTSCH, A.; EDELWEISS, N.; SANTOS, C. S. Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Databases. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 13., 2002, Aix-en-Provence. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 2002. p.524-533.

GALANTE, R.M.; EDELWEISS, N.; SANTOS, C.S. Change Management for Temporal Versioned Object-Oriented Database. In: WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, 2., 2002, Tampere. **Evolution and Change in Data Management**: proceedings. Berlin: Springer-Verlag, 2002.

GOLENDZINER, L.G. **Um Modelo de Versões para Bancos de Dados Orientados a Objetos**. 1995. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

GORALWALLA, I; ÖZSU, M.; SZAFRON, D. An Objetct-Oriented Framework for Temporal Data Models. In: ETZION,O.; JAJODIA,S.; SRIPADA, S. (Ed.). **Temporal Databases**: research and practice. Berlin: Springer, 1998. p. 1-35. (Lecture Notes in Computer Science, 1399).

GRANDI, F. A Relational Multi-Schema Data Model and Query Language for full Support of Schema Versioning. In: NATIONAL CONFERENCE ON ADVANCED DATABASE SYSTEMS, 2002, Firenze. **Advanced Database Systems**: proceedings. Italy: Stampa 2P, 2002. p. 323-336.

GRANDI, F.; MANDREOLI, F. **A Formal Model for Temporal Schema Versioning in Object-Oriented Databases**. January 15, 2002. (Time Center Technical Report, TR-68). Disponível em: <www.scienceindex.com>. Acesso em: 23 out. 2002.

GRANDI, F.; MANDREOLI, F. ODMG language extensions for generalized schema versioning support. In: THE INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, ECDM, 1., 1999, Paris. **Workshop on Evolution and Change in Data Management** proceedings. Berlin: Springer, 1999. p. 36-47. (Lecture Notes in Computer Science, 1727).

HÜBLER, P. N. **Definição de um Gerenciador para o Modelo de Dados Temporal TF-ORM**. 2000. 92p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

JENSEN, C.S. **Temporal Database Management**. 1999. Dr. Technical Thesis. Department of Computer Science, Aalborg University, Aalborg. Disponível em: <http://www.cs.auc.dk/~csj/thesis/>. Acesso em: 10 set. 2002.

JENSEN, C.S. et al. The Consensus Glossary of Temporal Database Concepts – February 1998 Version. In: ETZION, O.; JAJODIA, S.; SRIPADA, S. (Ed.). **Temporal Databases**: research and practice. Berlin: Springer, 1998. p. 367-405.

LERNER, B.S. A Model for Compound Type Changes Encountered in Schema Evolution. **ACM Transactions on Database Systems**, New York, v. 25, n.1, p. 83-127, Mar. 2000.

LIU, L. Maintaining Database Consistency in the Presence of Schema Evolution. In: MEERSMAN, R.; MARK, L. (Ed.). **Database Applications Semantics** [S.l]: Chapman E Hall, 1997. p. 549-571.

MANDREOLI, F. **Schema Versioning in Object-Oriented Databases.** 2001. 153 p. Tese (Doutorado em Ciência da Computação) – DEIS, Università degli Studi di Bologna, Italy.

MCKENZIE, L.E.J.; SNODGRASS, R.T. Evaluation of Relation Algebras Incorporating the Time Dimension in Databases. **ACM Computing Surveys**, New York, v. 23, n. 4, p. 501-543, Dec. 1991.

MOREIRA, V.P.; EDELWEISS, N. Schema Versioning: queries to a Generalized Temporal Database System. In: WORKSHOP ON SPATIO-TEMPORAL DATA MODELS AND LANGUAGES, 10., 1999, Florence. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 1999.

MOREIRA, V.P. **Consultas a Bancos de Dados temporais que Suportam Versionamento de Esquemas.** 1999. 140 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MORO, M.M. **Modelo Temporal de Versões**. 2001. 108 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MORO, M.M.; SAGGIORATO, S.M.; EDELWEISS, N.; SANTOS, C.S. Adding Time to an Object-Oriented Versions Model. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 12., 2001, Munich. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 2001-a.

MORO, M.M.; GELATTI, P.C.; GOMES, C.H.P.; ROSSETTI, L.L.F.; ZAUPA, A.P.; EDELWEISS, N.; SANTOS, C.S. **Linguagem de Consultas para o Modelo Temporal de Versões**. Porto Alegre: PPGC da UFRGS, 2001-b. 92f. (RP-308).

MORO, M.M.; ZAUPA, A.; EDELWEISS, N.; SANTOS, C.S. TVQL – Temporal Versioned Query Language. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 13., 2002, Aix-en-Provence. **Database and Expert Systems Applications**: proceedings. Berlin: Springer-Verlag, 2002.

RA, Y.G; RUNDENSTEINER, E. A. A Transparent Object-Oriented Schema Change Approach using View Evolution. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 11., 1995, Taiwan. **Proceedings…** Disponível em: <www.scienceindex.com>. Acesso em: 20 nov 2002.

RODDICK, J.F. **A Model for Temporal Inductive Inference and Schema Evolution in Relational Database Systems**.[S.l.]: Department of Computer Science and Computer Engineering, La Trobe University, 1994.

RODDICK, J.F. A survey of schema versioning issues for database systems. **Information and Software Technology**, [S.1.], v. 37, n.7, p. 383 - 393, 1996.

RODDICK, J.F et al. Evolution and Change in Data Management – Issues and Directions. In: **SIGMOD Record**, New York, v. 29, n. 1, p. 21-25, Mar. 2000.

RODDICK, J.F.; SNODGRASS, R.T. Schema Versioning In: SNODGRASS, R.T. et al. **The TSQL2 Temporal Query Language.** [S.l.:s.n.], 1995.

RODRÍGUEZ, L.; OGATA, H.; YANO, Y. A Temporal Versioned Object-Oriented Data Schema Model. **Information Sciences**, [S.l.], v. 114, n. 1-4, p. 281-300, Mar. 1999.

RODRÍGUEZ, L.; OGATA, H.; YANO, Y. An Access Mechanism for a Temporal Versioned Object-Oriented Database. In: **IEICE Trans. Inf. & Syst.**, [S.l.], v. E82D, n. 1, p. 128-135, Jan. 1999.

ROMA, A.B.S. **Um Modelo para Evolução de Esquemas em Bancos de Dados Orientados a Objetos Usando Versões e Operações Complexas**. 2000. 96p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ROMA, A.B.S.; JANTSCH, A.; GALANTE, R.M.; EDELWEISS, N.; SANTOS, C.S. Gerenciamento Temporal de Versões para Evolução de Esquemas em BDOO. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 16., 2001, Rio de Janeiro. **Anais…** Rio de Janeiro: UFRJ, 2001. p. 110-124.

SKJELLAUG, B. **Temporal Data:** time and object databases. Oslo: University of Oslo, Department of Informatics, 1997. (Research Report, 245).

SNODGRASS, R.T. **Developing Time – Oriented Database Applications in SQL**. San Francisco: Morgan Kaufmann, 2000.

TANSEL, C.G. et al. (Ed.). **Temporal Databases – theory, design and implementation**. Redwood City: Benjamin/Cummings, 1993.

WEI, H.-C.; ELMASRI, R. PMTV: Schema versioning for Bi-temporal Databases. In: INTERNATIONAL WORKSHOP ON TEMPORAL REPRESENTATION AND REASONING, 7., 2000. **Proceedings…** [S.l.]: IEEE, 2000.

ZANIOLO, C. et al. **Advanced Database Systems**. San Francisco: Morgan Kaufmann Publishers, 1997.

ZAUPA, A. P. **Suporte a Consultas no Ambiente Temporal de Versões**. 2002. 108p.Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.