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

Suporte a Consultas no Ambiente Temporal de Versões

por

AGLAÊ PEREIRA ZAUPA

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

Profa. Dra. Nina Edelweiss
Orientadora

Prof. Dr. Clesio Saraiva dos Santos
Co-orientador

Porto Alegre, agosto de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Zaupa, Aglaê Pereira
Suporte a Consultas no Ambiente Temporal de Versões / por Aglaê Pereira Zaupa. – Porto Alegre : PPGC da UFRGS, 2002.
108 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientadora: Edelweiss, Nina; Co-Orientador: Santos, Clesio Saraiva dos.

1. Banco de Dados Temporais. 2. Modelo de Versões. 3. Linguagem de Consulta. I. Edelweiss, Nina. II. Santos, Clesio Saraiva dos. III. Título.

Agradecimentos

Agradeço inicialmente a Deus, por permitir que eu atingisse mais este objetivo.

À professora Nina, uma excelente orientadora e uma pessoa formidável. Sempre com uma solução ou uma forma de me guiar à solução dos problemas que apareceram durante a realização deste trabalho. Dona de uma sensibilidade incrível, sempre soube me conduzir e me tranquilizar, mesmo quando eu desanimava ou quando meu nível de estresse estava elevado. Professora Nina, muito obrigada!

Ao meu co-orientador professor Clesio por me ajudar sempre que necessário.

À Mirella, pela ajuda grandiosa, paciência e disposição para solucionar minhas dúvidas. Sua contribuição foi muito importante para o enriquecimento deste trabalho. Muito obrigada!

Às amigas que conquistei em Porto Alegre: Adriana, Anelise e Sílvia. Pessoas que sempre tinham uma palavra para levantar o meu astral quando alguma coisa não acontecia como eu havia previsto. Afinal, "no final tudo dá certo. Se não der, é porque ainda não é o final", não é mesmo?

A todos os professores e funcionários do PPGC da UFRGS que direta ou indiretamente me ajudaram e me acompanharam durante todo esse curso.

Ao pessoal de Londrina e de Presidente Prudente, pelos momentos que passamos juntos. Sabemos o quanto foi difícil, mas foram bons momentos que certamente lembraremos para sempre. Especialmente, a vocês companheiros de estrada (no sentido mais amplo da palavra): Cássia, Melissa e Silvio, pelas noites, sábados e domingos que passamos estudando, fazendo trabalhos e planos para o futuro.

Agradeço também à UNOESTE, Universidade do Oeste Paulista, pelo apoio financeiro.

Aos funcionários do centro de processamento de dados e da faculdade de informática da UNOESTE pelo apoio e compreensão. Especialmente para: César, Danilo, Elias, Eliezer, Luisa, Toninho, Marcelo, Mário e Moacir.

Ao meu Pai Adilon e à minha Mãe Delci, por sempre me apoiarem em tudo o que fiz. Um apoio incondicional, capaz de suportar a minha ausência e não medir esforços para ajudar a atingir meus objetivos. Meus heróis, eu agradeço do fundo do meu coração. Amo vocês!

Ao meu irmão André por entender minha ausência nas fases mais difíceis e complicadas de sua vida, onde talvez a irmã mais velha fosse importante.

Por último e principalmente, agradeço ao meu amigo e grande amor Leandro, pelo apoio, companheirismo, preocupação e pelo sacrifício que fizemos. Por compreender minha ausência e ainda assim ter forças para me ajudar e me incentivar. Sem você ao meu lado, eu não teria chegado até aqui. Leandro, obrigada. Eu te amo!

Sumário

Lista de Abreviaturas

ANSI	American National Standards Institute
BD	Banco de Dados
BDR	Banco de Dados Relacional
BDOO	Banco de Dados Orientado a Objetos
BDT	Banco de Dados Temporal
BNF	Backus-Naur Form
CAD	Computer Aided Design
CASE	Computer Aided Software Engineering
OID	Object Identifier
OO	Orientado a Objetos
SCM	Software Configuration Management
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDOO	Sistema de Gerenciamento de Banco de Dados Orientado a Objetos
SQL	Structured Query Language
TVM	Temporal Versions Model
TVQL	Temporal Versioned Query Language
XML	Extensible Markup Language

Lista de Figuras

Lista de Tabelas

Resumo

O Modelo Temporal de Versões (TVM-*Temporal Versions Model*) foi proposto com base na união de um modelo de versões com informações temporais. Esse modelo permite o armazenamento de alternativas de projeto, o armazenamento da história dos dados em evolução, bem como a reconstrução do estado da base em qualquer data passada, sem o uso de operações complexas de *backup* e *recovery*.

Para realizar consultas nesse modelo foi definida uma linguagem de consulta, a TVQL (*Temporal Versioned Query Language*). Além das consultas básicas realizadas pela linguagem padrão SQL, a TVQL permite novas consultas que retornam valores específicos das características de tempo e versões, estabelecendo um comportamento o mais homogêneo possível para elementos normais e temporais versionados.

O objetivo principal deste trabalho é possibilitar a realização de consultas TVQL em um banco de dados convencional. Nesse contexto, o mapeamento da TVQL é implementado através da tradução de todas as propriedades e funções definidas na TVQL para SQL. Para que isso seja possível é necessário que os dados também estejam nesse banco de dados. Então, faz-se necessário o mapeamento das classes da hierarquia do TVM, bem como das classes da aplicação do usuário, para o banco de dados.

Adicionalmente, é implementado um protótipo de uma interface de consultas realizadas em TVQL, para testar o funcionamento tanto da TVQL como do seu mapeamento.

Palavras-chave: modelo de dados temporal, modelo de versões, banco de dados relacional, linguagem de consulta

TITLE: “QUERIES SUPPORT IN A TEMPORAL VERSIONS ENVIRONMENT”

Abstract

The Temporal Versions Model (TVM) was proposed based on the union of a versions model with temporal information. This model allows the storage of design alternatives, together with the storage of the data evolution history. This way, the reconstruction of a past state of the database, corresponding to a past date, is also possible, without the use of complex operations of backup and recovery.

To solve queries when using this model, a query language was defined, called TVQL (Temporal Versioned Query Language). Besides the basic queries carried through by the standard SQL language, TVQL allows queries that retrieves specific values of temporal and versions characteristics, establishing an homogeneous behavior for normal and temporal versioned elements.

The main purpose of this work is to execute TVQL queries using a conventional database. In this context, first of all, TVM was mapped to a conventional database. This is achieved mapping the TVM hierarchy classes, together with the user-defined classes, to the used database. The mapping of TVQL is then implemented, translating all the properties and functions defined in TVQL to SQL.

Additionally, a prototype of a TVQL query interface carried through in TVQL is implemented, in order to test both the functioning of TVQL and its mapping.

Keywords: temporal data model, versions model, relational database, query language

1 Introdução

1.1 Motivação

A recuperação e a manipulação de informações armazenadas em um banco de dados é fundamental. Para isso, os modelos de dados devem apresentar uma linguagem de recuperação de informações.

Um modelo de dados temporal permite a representação de aspectos estáticos e dinâmicos de uma aplicação, bem como sua evolução temporal [SNO 2000][TAN 93][ZAN 97]. Bancos de Dados Temporais (BDT) armazenam todos os estados de uma aplicação (presente, passado e futuro), mantendo sua evolução com o passar do tempo. As informações temporais são associadas implicitamente aos dados, correspondendo ao tempo de validade (tempo no qual a informação é valida no mundo real) e/ou ao tempo de transação (tempo no qual a informação foi inserida no banco de dados).

Em banco de dados convencionais existe somente a representação do estado presente, ou seja, quando o mundo real muda, os novos valores são incorporados ao banco de dados (BD) substituindo os valores antigos. No entanto, em BDT as propriedades temporais são atualizadas através da inserção de um novo fato no BD [EDE 2000].

A preocupação com a representação do tempo na modelagem de dados levou à proposta de vários modelos de dados temporais. Avaliando de forma concisa as publicações na área de banco de dados temporal, fica nítida uma evolução de assuntos pesquisados. Primeiro vieram os modelos de dados temporais com inúmeros detalhes e riqueza semântica. Logo após, surgiram as linguagens de consultas temporais seguidas pelas lógicas temporais, regras de integridade e rótulos temporais. Atualmente, as publicações demonstram pesquisas sobre questões bem mais específicas (otimização de consultas com dados duplicados, multigranularidade, consulta para OLAP, indeterminância, etc), ou sobre um domínio específico (multimídia, dados semi-estruturados, banco de dados ativos, etc). Entre os artigos que retratam essa situação estão as referências [BOW 2001][COW 2000][EDE 2000][GOR 2001][LIW 2001][MIR 2000][OLI 2001][ROD 2000][ROS 93][SAR 90][SNO 2000][SNO 96][VAI 2001].

Os comandos para recuperação de informações em BDT constituem um tipo especial de consultas, onde o fator tempo pode ser envolvido para recuperar valores de propriedades cujo domínio é temporal; para se referir a um determinado instante ou intervalo temporal; e para recuperar valores com base em restrições temporais. Além disso, as possíveis consultas temporais não dependem somente da especificação da informação buscada, mas também do tipo de banco de dados implementado, do modelo de dados utilizado e da história considerada.

Ainda no contexto de evolução temporal, diversas aplicações necessitam de mecanismos de suporte ao processo de desenvolvimento evolutivo. Além de armazenar diferentes estágios de uma entidade em tempos distintos, é necessário armazenar sob diferentes pontos de vista compondo diferentes versões de projeto.

Historicamente, as primeiras pesquisas relacionadas a versões se encontram nas áreas de CAD (*Computer Aided Design*), CASE (*Computer Aided Software Engineering*) e SCM (*Software Configuration Management*) [ABD 97][AGR 91][AND

97][CON 98][GOL 95][KAT 90][KIM 89][WAG 91][WUU 93]. Posteriormente, esse conceito foi se estendendo para outros domínios de aplicação. Atualmente, as pesquisas também se concentram em aspectos mais específicos, como gerenciamento de versões de documentos [CHI 2001][NOR 98][SOA 95] e XML [CHI 2002][CHI 97], versões e evolução de esquema [LAU 97][GAL 2002] e versões em projetos concorrentes [ALK 2001].

Apesar dos modelos de dados versionados armazenarem alternativas de projeto, algumas modificações podem ter sido realizadas, que de alguma forma influíram no desenvolvimento geral, impossibilitando o acesso posterior aos valores originais. Para resolver esse problema seria necessário um modelo que suportasse os aspectos de versões bem como os temporais.

Juntando esses conceitos, alguns trabalhos propõem a união dos modelos temporais e de versões [MOR 2001a][ROL 99][WUU 93]. No trabalho de Moro, particularmente, são armazenadas as versões de um objeto, e para cada versão, o histórico das alterações feitas nos valores de seus atributos e relacionamentos dinâmicos [MOR 2001a]. Esse modelo denomina-se TVM (*Temporal Versions Model*).

O TVM faz parte de um projeto que implementa um ambiente integrado para especificação de classes, versionamento de objetos, gerenciamento de objetos, consultas e visualização. Possui as características únicas de herança por extensão e ordem temporal ramificada. Além disso, permite classes sem tempo e versões entre as classes temporais versionadas.

A partir da definição do TVM, surgiu a necessidade de realizar consultas sobre os dados do modelo. Foi então definida a TVQL (*Temporal Versioned Query Language*), uma linguagem para consultas às versões considerando também o aspecto temporal do TVM [MOR 2001b].

A TVQL é baseada em SQL, da qual mantêm as características gerais. Adicionalmente, foi feita a definição de algumas construções, visando a análise das informações, permitindo assim a recuperação de todas as informações deste modelo, envolvendo versões e histórico. Esta dissertação está baseada no modelo TVM e em sua linguagem de consulta, a TVQL.

Como não existe um SGBD (Sistema de Gerenciamento de Banco de Dados) próprio para o TVM, este precisa ser implementado sobre um BD convencional. Para posteriormente poder consultar as informações armazenadas através da linguagem de consulta do TVM – a TVQL, é necessário mapear esta última para a linguagem de consulta do BD utilizado na implementação. Deste modo é possível recuperar as informações específicas do modelo (de tempo e de versões dos dados), como também as informações estáticas.

1.2 Objetivo

O objetivo principal desse trabalho é propor o mapeamento da linguagem de consulta do TVM para um banco de dados relacional convencional. Basicamente, esse processo une as duas partes do mapeamento do modelo (representação da hierarquia de classes do modelo e a implementação das classes da aplicação) com o mapeamento das funções e propriedades da TVQL para SQL.

O banco de dados utilizado no mapeamento é o DB2, o que leva à necessidade de mapear as classes do modelo e de aplicação para esse BD. A escolha do banco de

dados DB2 se deve ao fato de ser o mais próximo ao padrão SQL-92 e pelo suporte aos tipos de dados temporais [SNO 2000].

Adicionalmente, foi implementado o protótipo de uma ferramenta para consultas em TVQL, mostrando o funcionamento do mapeamento das consultas para SQL.

1.3 Organização do texto

O restante do texto está organizado como segue:

- o capítulo 2 apresenta conceitos básicos sobre modelos temporais, modelos com suporte a versões e linguagens de consulta. Além disso, mostra as características do TVM;
- o capítulo 3 aborda a definição da linguagem de consulta do TVM, a TVQL, destacando suas principais características;
- o capítulo 4 engloba a definição do mapeamento do modelo TVM para um SGBD convencional. É abordado tanto o mapeamento das classes de aplicação como da hierarquia de classes do modelo;
- o capítulo 5 mostra o mapeamento da TVQL para o DB2, sendo todas as funções e propriedades traduzidas para a SQL padrão;
- o capítulo 6 apresenta o protótipo implementado, e sua interação com o Ambiente Temporal de Versões. Um estudo de caso ilustra a sua utilização;
- o capítulo 7 apresenta as conclusões obtidas sobre esta pesquisa, bem como as sugestões para trabalhos futuros;
- o Anexo 1 apresenta a BNF da TVQL;
- o Anexo 2 contém o *script* de criação das classes da hierarquia do TVM, mapeadas para o DB2;
- o Anexo 3 mostra o *script* utilizado para gerar as tabelas mapeadas do TVM referente às classes de aplicação do modelo apresentado no estudo de caso.

2 Base Conceitual

Este capítulo resume os principais conceitos relacionados aos aspectos temporais bem como de versões, abrangendo os modelos de dados e as linguagens de consulta aos dados que suportem essas características. Além disso, são relatados alguns trabalhos que vêm sendo desenvolvidos envolvendo os aspectos temporais e de versionamento. Especificamente, é detalhado o Modelo Temporal de Versões, cuja linguagem de consulta é apresentada no próximo capítulo.

2.1 Modelo de Dados Temporais

Bancos de dados armazenam informações do mundo real. O fato de que as informações evoluem com o tempo levou ao desenvolvimento de banco de dados temporais [EDE 98][TAN 93]. Esses bancos de dados têm o objetivo de integrar em um sistema as informações referentes ao passado, presente e futuro, armazenando uniformemente os estados de um objeto com relação à sua evolução no decorrer do tempo. Informações temporais são associadas implicitamente aos dados (tempo de transação e/ou tempo de validade) [JEN 98].

Informações, tais como valores temporais, restrições temporais e características de evolução temporal, estão presentes em grande número de aplicações do mundo real. Na coleta dos requisitos de um sistema de informação, devem ser especificados os requisitos temporais da aplicação em questão e o método utilizado na especificação deve permitir que todos os aspectos temporais sejam representados.

Os modelos de dados tradicionais apresentam duas dimensões: (i) representando as instâncias dos dados (linhas de uma tabela), e (ii) representando os atributos de cada instância (colunas desta tabela). Cada atributo de uma instância apresenta um só valor e cada alteração desse valor implica na perda do anterior. Para os modelos temporais é acrescentada mais uma dimensão aos modelos tradicionais, chamada dimensão temporal, a qual associa uma informação temporal a cada valor. Nesse caso, se o valor de um atributo for alterado, o valor anterior não é removido do BD [EDE 98].

A utilização de um modelo de dados temporal para especificação de uma aplicação não implica, necessariamente, na utilização de um SGBD específico para o modelo. Bancos de dados comerciais podem ser utilizados se existir um mapeamento adequado entre o modelo temporal e o banco de dados utilizado. Um banco de dados temporal (BDT) pode ser implementado sobre um banco de dados relacional, orientado a objetos, objeto-relacional e outros. Em cada um deles devem ser preservadas as características individuais, bem como as regras que regem cada banco de dados (BD).

Considerando, por exemplo, a classe `Pessoa` que possui entre outros, o atributo temporal `nome`, esse deve armazenar os rótulos temporais durante os processos de inserção, alteração e exclusão lógica[1]. Sendo assim, algumas considerações devem ser feitas durante esses processos:

- o usuário deve fornecer o valor da propriedade e os tempos de validade inicial e, opcionalmente, o tempo de validade final da informação que está

[1] Em banco de dados temporal, uma exclusão é sempre lógica, pois os valores não são excluídos – seus tempos de validade são encerrados.

sendo inserida. Se a validade final da propriedade não é informada, é armazenado o valor null e, nesse caso, o valor da propriedade é considerado válido até que outra informação seja definida ou o objeto seja excluído;

- os tempos de transação são fornecidos pelo SGBD. Na inserção somente o tempo de transação é registrado. O tempo de transação final é registrado quando uma nova instância para o objeto é inserida, atualizada ou excluída logicamente da base de dados;
- nenhum atributo é fisicamente modificado, exceto o que tem o tempo de transação final em aberto [ELM 2000].

Para demonstrar o processo de atualização, a tabela 2.1 apresenta algumas operações realizadas sobre a propriedade temporal `nome` da classe `pessoa`. No dia 1 de março de 2000 a pessoa de nome "Ana Souza" teve seus dados inseridos no banco de dados e no mesmo dia passaram a ser válidos. Este nome teve seu tempo de validade encerrado no dia 31 de março de 2001 e no dia 01/04/2001 um novo valor foi atribuído ("Ana Silva") passando a ser válido na mesma data. Este nome ficou válido até 25 de junho de 2002, pois no dia 12 de junho foi inserida uma nova informação ao nome da pessoa e esta passou a se chamar "Ana S. Souza", no entanto esta informação passaria a ser válida somente no dia 26 de junho de 2002.

TABELA 2.1 – Exemplo de rótulos temporais

valor	itTime	ftTime	lvTime	fvTime
Ana Souza	01/03/2000	31/03/2001	01/03/2000	
Ana Souza	01/04/2001		01/03/2001	31/03/2001
Ana Silva	01/04/2001	11/06/2002	01/04/2001	
Ana Silva	12/06/2002		01/04/2002	25/06/2002
Ana S. Souza	12/06/2002		26/06/2002	

A consistência de um BD é assegurada através da definição de regras, as quais controlam a integridade e a evolução dos dados. Conforme o tipo do BD utilizado, as regras são tratadas de formas diferenciadas, pois cada um possui características distintas e enfoques diferentes [MOV 99].

A implementação de bancos de dados temporais em sistemas gerenciadores de banco de dados (SGBD) convencionais deve-se à inexistência de um SGBD totalmente temporal. Nesse caso, são feitos mapeamentos apropriados para representar modelos temporais necessitando uma estratégia específica para sua representação, para que o gerenciamento dos dados históricos seja independente da intervenção do usuário.

Algumas implementações de bancos de dados temporais em SGBDs relacionais e orientados a objetos são propostas nas referências [EDE 2000][CAV 95][HEL 75] [JEN 99][SIM 98][TAN 93]. Essas implementações, geralmente, estão baseadas em modelos de dados já publicados na literatura.

Nos modelos relacionais temporais, a associação de tempo é feita, na maioria das vezes, através de atributos adicionais denominados rótulos temporais. As informações temporais podem ser associadas nos níveis de banco de dados, relacionamentos e tuplas [HÜB 2000].

Em modelos orientados a objetos, o aspecto temporal tem a finalidade de representar todos os estados assumidos pelo objeto durante sua existência. Para isso, o

modelo deve permitir a representação do comportamento que o objeto pode apresentar em sua evolução. Sendo assim é preciso considerar alguns aspectos, tais como:

- a forma utilizada para a representação temporal - a qual pode ser feita nos objetos ou nos atributos. No último caso é possível que as propriedades de um objeto variem assincronamente com o tempo [TAN 93];
- a forma como ocorrerá a alteração durante a evolução (tanto no nível de classes como no de objetos) - contínua, discreta ou por degraus;
- a possibilidade de migração de objetos entre classes – esta quando permitida, normalmente é restrita à migração entre classes e subclasses, pois a migração genérica entre quaisquer duas classes apresenta restrições muito fortes;
- a possível evolução dos esquemas conceituais (criação de classes, alterações na hierarquia de generalização/especialização, etc.) [EDE 98].

Com relação à representação de tempo, os BDs podem ser classificados em quatro tipos: (i) BD instantâneos, os quais armazenam somente os valores em um determinado instante de tempo, sendo que a cada modificação no valor de uma propriedade, o valor anteriormente armazenado é perdido permanecendo somente o último valor; (ii) de tempo de transação, onde o SGBD adiciona automaticamente o tempo de transação a cada valor definido, sob a forma de um rótulo temporal (*timestamp*), ficando armazenados no BD todos os valores associados ao tempo de transação; (iii) de tempo de validade, nos quais o tempo deve ser fornecido pelo usuário e está associado a cada informação com relação ao seu tempo de validade no mundo real; e (iv) os BD bitemporais que associam os tempos de transação e de validade a cada informação, constituindo a forma mais completa de armazenar informações relacionadas a tempo, pois tanto a história das transação realizadas, como a história da validade dos dados, são armazenadas [EDE 98].

2.2 Modelo de Dados com suporte a Versões

Com o passar do tempo a estrutura do banco de dados pode sofrer alterações por vários motivos. Nesses casos, há a necessidade de guardar mais de um estado para um objeto e, para satisfazer esse requisito, o conceito de versões pode ser incorporado à semântica do sistema. Uma versão corresponde a um estado identificável de um objeto e deve ser tratada uniformemente em um modelo de dados [GOL 95].

Existem na literatura vários estudos sobre versões em modelos de dados relacionais [DAD 84] e orientados a objetos [AGR 91][BEE 88][BJÖ 89][CON 98] [GOL 95][KAT 90][KIM 89].

Em modelos orientados a objetos, um objeto pode ser representado através de versões. Uma versão é a descrição de um objeto em um determinado período de tempo, ou sob um ponto de vista. Nesses modelos, uma versão é um objeto de primeira classe, possuindo seu próprio identificador (*Object Identifier* - OID), o que permite que seja diretamente manipulada ou consultada como qualquer outro objeto. Objetos versionados são objetos que apresentam versões [GOL 95]. Os objetos que possuem versões são chamados de objetos versionados. Um objeto versionado pode ter mais de uma versão, sendo estas organizadas formando um grafo acíclico dirigido, representando a ordem de derivação.

Como o aspecto de versionamento está associado aos objetos, um objeto não versionado pode passar a ser versionado dinamicamente. Nesse processo, esse objeto não versionado passa a ser a primeira versão do objeto versionado e o antecessor da nova versão.

Todo objeto versionado possui uma versão corrente que é gerenciada pelo sistema, sendo esta atualizada sempre que novas versões forem criadas ou no caso do usuário especificar outra versão como sendo a versão corrente do objeto versionado. Nesse caso, esta versão permanece fixa.

O modelo proposto por Golendziner [GOL 95] é um modelo de dados orientados a objetos que suporta versões. Esse modelo permite a definição e a manipulação de objetos, versões e configurações, possibilitando a navegação através da hierarquia de herança e permitindo manter transparente a manipulação de versões, quando necessário. O trabalho de Moro et al. [Moro 2001a] propõe um modelo que une os conceitos de tempo ao modelo de versões proposto por Golendziner, permitindo representar toda a história e a evolução dos dados da aplicação, bem como manter as alternativas de projeto. Por representar o modelo de dados base deste estudo, o TVM é mostrado com mais detalhes na próxima seção.

2.3 Modelo Temporal de Versões (TVM)

Diversas aplicações do mundo real necessitam armazenar o histórico dos seus dados e suas representações, podendo ser revisões ou alternativas criadas, ou seja, versões. Bancos de dados tradicionais não implementam de forma natural tais conceitos, sendo necessária a extensão dos mesmos para um modelo que suporte dados temporais e versionados.

Deste modo, o Modelo Temporal de Versões (TVM - *Temporal Versions Model*) [MOR 2001a] foi definido para permitir o armazenamento das versões de objetos e, para cada versão, o histórico de suas propriedades dinâmicas e valores dos relacionamentos. Contemplando esses requisitos, a seguir são apresentadas as principais características do TVM. Mais detalhes sobre o TVM podem ser encontrados em [MOR 2001a]. A linguagem de consulta mapeada no presente estudo foi definida com base nesse modelo.

2.3.1 Características do modelo

O TVM (*Temporal Versions Model*) corresponde a uma extensão temporal ao modelo de versões de Golendziner. Esse modelo considera o gerenciamento das versões dos objetos da aplicação, suportando a representação das informações dependentes de tempo e seqüenciamento, conforme definido pelo usuário da aplicação. Com isso, a associação da dimensão temporal habilita armazenar todo o histórico dessa aplicação e a evolução dos valores dos atributos e relacionamentos das instâncias. No entanto, o usuário deve definir para quais elementos ele deseja que o histórico seja armazenado, para não sobrecarregar a capacidade de armazenamento e o desempenho do banco de dados.

No TVM, o tempo é associado a objetos, versões, atributos e relacionamentos, proporcionando uma modelagem mais flexível. Um objeto tem uma linha de tempo para cada versão. Esse modelo possibilita duas ordens no tempo: (i) tempo ramificado para um objeto, devido às diferentes linhas de tempo de cada versão que são originadas da linha do objeto; e (ii) tempo linear para cada versão. A variação temporal é discreta, e a temporalidade é representada no modelo através do rótulo de tempo elemento temporal

(conjunto de intervalos temporais), bitemporal (tempos de transação e validade) e implícito. No TVM, o usuário pode especificar classes normais, sem tempo e versão, além das classes temporais versionadas, permitindo a integração com especificações existentes.

Os objetos possuem um atributo temporal pré-definido `alive` que representa seu tempo de vida. Esse atributo recebe o valor `true` no momento da criação do objeto, armazenando também seu tempo de validade inicial. O tempo de validade final recebe o tempo de transação no momento da exclusão lógica e seu valor passa para `false`.

2.3.2 Hierarquia de Classes do TVM

A figura 2.1 ilustra a hierarquia de classes base definida para o TVM.

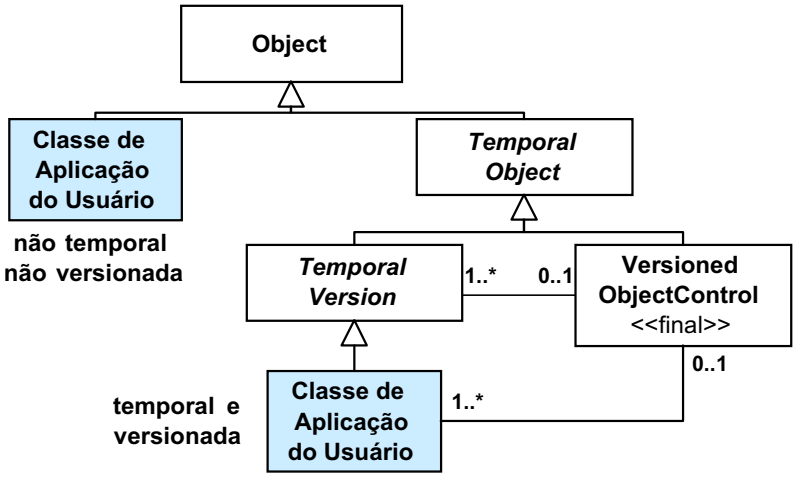

FIGURA 2.1 – Hierarquia definida para o modelo TVM

A partir dessa hierarquia é possível especificar dois tipos de classes da aplicação: classe de aplicação **não temporal e não versionada**, e classe de aplicação **temporal e versionada**. A primeira pode ser usada para representar classes referentes a tabelas comuns de uma base de dados já existente ou classes auxiliares onde os aspectos de versões e tempo não são necessários. No segundo tipo de classe, suas instâncias, podem ser objetos não versionados, versionados ou ainda versões.

2.3.3 Características de Tempo

Considerando que o TVM é um modelo bitemporal, todas as propriedades temporais têm associadas os rótulos pré-definidos `ivTime`, `fvTime`, `itTime` e `ftTime`, que representam respectivamente, os tempos de validade inicial e final, e os tempos de transação inicial e final.

Regras de integridade temporal foram estabelecidas para garantir que o rótulo de tempo associado às versões esteja 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.

2.3.4 Gerenciamento de Versões

As versões em uma determinada aplicação representam as alternativas de projeto e a evolução dos seus dados. Cada versão possui um estado que reflete seu estágio de desenvolvimento e/ou consistência durante seu tempo de vida. Uma versão pode passar por alguns estados, onde cada estado define o conjunto de operações que podem ser

aplicadas sobre uma versão. As transições entre os estados e os eventos que causam essas transições são representadas pela figura 2.2.

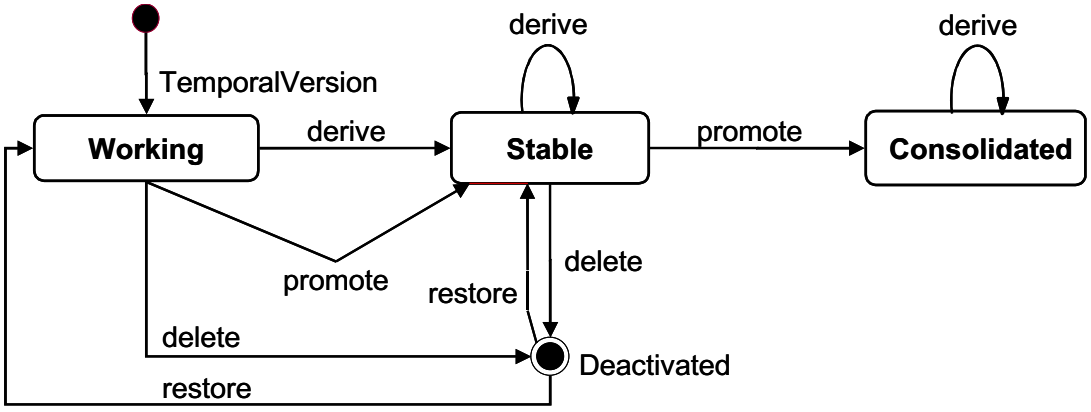

FIGURA 2.2 – Diagrama de estados de uma versão

Quando criada, a versão está no estado `working`, podendo servir como base de uma derivação, ser promovida para `stable`, alterada, consultada ou excluída logicamente. Nesse último caso, assume o estado `deactivated`.

No estado `stable`, a versão também pode servir como base para uma derivação, ser promovida para `consolidated`, consultada, compartilhada por outros usuários, excluída desde que não possua versões sucessoras, mas não pode mais ser alterada.

Quando uma versão passa para o estado `consolidated`, pode servir como base para derivação, consultada e compartilhada por outros usuários, não podendo mais ser alterada nem excluída.

Finalmente, no estado `deactivated` a versão pode ser apenas consultada e restaurada.

Na hierarquia de classes do TVM, o controle das versões é feito pela classe `VersionedObjectControl`, sendo que cada versão ou objeto versionado tem um relacionamento estabelecido com essa classe. Além disso, na hierarquia do TVM a mudança de uma classe, de não versionada para versionada, deve ser feita pelo usuário em tempo de projeto. No entanto, é importante considerar que é possível que um objeto de uma classe temporal versionada não tenha versão associada.

A figura 2.3 ilustra a forma como os objetos de uma classe temporal versionada são instanciados. Apresenta os objetos versionados `ObjVers1`, `ObjVers2` e `ObjVers3`, sendo que somente o `ObjVers1` possui versões associadas.

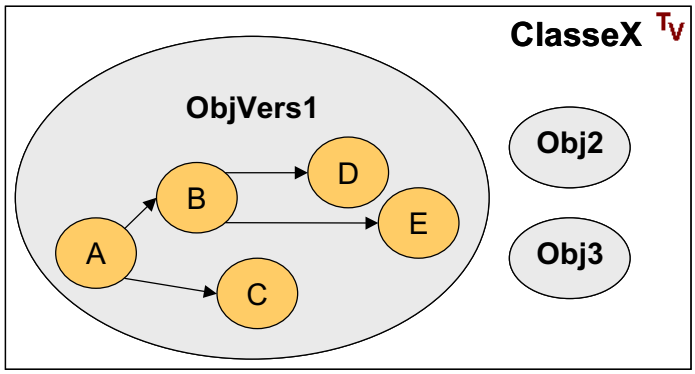

FIGURA 2.3 – Objetos versionados e não versionados de uma classe TVM

Outra característica relevante do TVM é a definição do **relacionamento de herança por extensão**, onde versões são admitidas em vários níveis da hierarquia de herança. Com isso, pode-se estabelecer correspondências entre versões de um objeto em uma classe e versões de seu objeto ascendente na superclasse. A correspondência estabelece uma restrição de integridade, que é especificada quando da definição do relacionamento de herança entre uma classe e sua superclasse.

Em resumo, o recurso de relacionamento de herança por extensão permite que um objeto possa ser desenvolvido em um determinado nível de abstração e posteriormente detalhado nos níveis inferiores da hierarquia. Desse modo, a modelagem das entidades do mundo real pode ser feita em vários níveis, projetando características de um objeto em uma camada de cada vez.

2.4 Linguagens de Consulta

É imprescindível que esteja disponível uma linguagem de consulta quando um BD é utilizado. Com relação às linguagens de consultas temporais, essas devem possibilitar a recuperação de informações armazenadas no banco de dados, independentemente de serem temporais ou não. Da mesma forma, as linguagens de consultas para versões possuem um papel importante, proporcionando operações para consultar a estrutura de versões e seus relacionamentos, além de obter informações sobre a estrutura completa das versões ou sobre cada versão.

Existem na literatura várias propostas para definição de linguagens de consulta para modelos de dados temporais. Já para modelos com suporte a versões existe um número significativamente inferior.

No contexto de versões, as linguagens de consultas são estendidas por algumas primitivas que permitem a manipulação de versões. Entretanto, do ponto de vista operacional, a navegação em um banco de dados que contem versões de objetos não difere muito da navegação sem versões. Em conseqüência da falta de semântica para versionamento, a mais expressiva linguagem de consulta é reduzida e uma grande parte do potencial da informação contida na base de dados não pode ser usada [ABD 97].

2.4.1 Linguagens de Consulta para Tempo

Consultas temporais constituem um tipo especial de consulta. Elas referem-se às propriedades temporais, com acesso a informações armazenadas ao longo do tempo. Uma linguagem de consulta temporal deve possibilitar a recuperação de informações armazenadas no banco de dados, independentemente de serem temporais ou não.

As consultas temporais permitem: (i) fornecer valores de propriedades cujo domínio é temporal; (ii) referir-se a um determinado instante ou intervalo temporal; (iii) recuperar valores com base em restrições de tempo; e (iv) fornecer informações temporais, como datas e intervalos. Para isso as linguagens de consultas temporais devem manipular a dimensão temporal e ter capacidade de dedução sobre o tempo com base nas informações temporais armazenadas. Isto é possível através da utilização de lógica temporal, a qual permite inferências de valores não explicitamente armazenados [EDE 98].

Em BDs instantâneos não é possível realizar consultas temporais, pois esses não possuem suporte para informações temporais. Já em BDs de tempo de transação podem ser recuperados valores definidos em tempos passados e atuais. Nos BDs de tempo de

validade podem ser feitas consultas a valores válidos no passado, presente e futuro. Por último, os BDs bitemporais associam as características dos BDs de tempo de transação e os de tempo de validade. Sendo assim, as consultas podem ser realizadas a informações do passado, presente e futuro, levando em consideração os tempos de transação e de validade.

A TQuel é uma extensão mínima (sintática e semanticamente) a Quel [HEL 75], a linguagem de consulta para o BD Ingres. O modelo de dados dessa linguagem é um modelo temporal relacional denominado TRDM (*Temporal Relational Data Model*). A TQuel suporta os tempos de validade e de transação, agregados, tempo de validade indeterminado (onde não se sabe exatamente quando um evento ocorreu), modificação da base de dados e evolução do esquema [TAN 93].

Na TQuel, a cláusula `WHEN` é adicionada à semântica da linguagem proporcionando a seleção por tempo de validade [SNO 87]. Os predicados temporais podem ser divididos em dois grupos: (i) os predicados que consistem em expressões de intervalo, em expressões de evento, e nos operadores temporal de predicado (`OVERLAP`, `PRECEDE` e `EQUAL`), que são sobrecarregados para o intervalo e a comparação do tempo do evento; (ii) as expressões booleanas que consistem em predicados do primeiro grupo e dos operadores lógicos.

A linguagem de consulta do modelo HDBMS (*Historical Database Managment System*) é chamada HSQL [SAR 90]. Essa linguagem é uma extensão da SQL com suporte a operações temporais, tais como comparações entre instantes de tempo, comparações entre intervalos e operações sobre intervalos. Apresenta também algumas funções para manipulação temporal como a extração de intervalos de tempo de tuplas e medida do tempo decorrido entre dois instantes. Tempos de diferentes granularidades podem ser comparados.

Na HSQL, o `FROM` e `TO` são operadores de extração que recuperam o início e o fim de um intervalo, respectivamente. O operador `INTERVAL` representa o rótulo temporal do intervalo de uma tupla em uma relação do estado. O operador `AT` indica a associação de tempo do evento com uma tupla de uma relação do evento [SAR 90].

O OODAPLEX [WUU 93] é um modelo orientado a objetos baseado no modelo funcional DAPLEX. O modelo OODAPLEX apresenta as principais características de um modelo orientado a objetos: identidade de objeto, encapsulamento, tipos de objetos complexos, herança e polimorfismo.

A linguagem de consulta do OODAPLEX permite a recuperação e a atualização de objetos individuais e de agregados de objetos, a recuperação e a navegação associativa, a manipulação de objetos existentes e a criação de novos objetos. Pode ser utilizada para resolver consultas temporais e não temporais.

As consultas são definidas através de expressões que denotam objetos. A expressão mais simples é uma variável ou uma constante, e o seu valor é o objeto denotado por esta variável ou constante. Expressões complexas utilizam a aplicação de funções ou a formação de agregados. A aplicação de funções é a operação primitiva da linguagem de consulta do OODAPLEX. Expressões de formação de agregados incluem expressões de conjuntos de tuplas e de multiconjuntos.

O TF-ORM (*Temporal Functionality in Objects with Roles Model*) é um modelo de dados temporal orientado a objetos que utiliza o conceito de papéis para representar os diferentes comportamentos dos objetos [EDE 98]. A linguagem de consulta TF-ORM

é baseada em SQL e apresenta algumas influências de TQuel [SNO 87]. Apresenta extensões para suportar os aspectos temporais e para isso, possui mais uma cláusula especial (determina a história do banco de dados que deve ser considerada) e um conjunto de operadores temporais e predicados que podem ser utilizados para recuperar informações em determinados pontos no tempo.

A TSQL é uma linguagem de consulta temporal baseada na SQL e desenvolvida para o modelo relacional *Temporal Relational Model* (TRM). Nesse modelo cada esquema relacional temporal possui dois atributos temporais (`tempo_de_início` e `tempo_de_fim`) que correspondem ao limite inferior e superior de um intervalo. As relações temporais representam o tempo de validade e as propriedades temporais são definidas para representar os tempos de transação e os definidos pelo usuário.

Na TSQL foram introduzidos vários novos componentes de sintaxe e semântica para suportar os aspectos temporais. Foi definida a cláusula `WHEN` que é semelhante à cláusula `WHERE` da SQL, mas com o objetivo de especificar os relacionamentos temporais das tuplas participantes na derivação. A recuperação de rótulos temporais é especificada na cláusula `SELECT` através dos operadores `TIME-START` e `TIME-END`.

Mesmo apresentando vários novos componentes que não possuem um equivalente em SQL (ordenação temporal, janela móvel, as cláusulas `WHEN` de `TIME-SLICE`), a TSQL provê uma poderosa capacidade de processamento de consultas com uma sintaxe simples e amigável, capaz de formular uma gama enorme de consultas relacionadas ao aspecto temporal.

Baseada na SQL, a TSQL2 [SNO 95] foi definida para o modelo temporal relacional BCDM – *Bitemporal Conceptual Data Model*. Esse modelo foi definido considerando o tempo como linear e discreto. Ambos os tempos são suportados - de transação e de validade, sendo assim um BD bitemporal.

O modelo BCDM suporta três linhas de tempo: tempo definido pelo usuário, tempo de validade e tempo de transação. Além disso, assume que um instante na linha do tempo é muito menor que um *chronon*, que é a menor entidade que um rótulo temporal pode representar [SNO 95]. A associação temporal é implícita, sendo o rótulo temporal associado às tuplas onde, a cada tupla é associado um subconjunto arbitrário do domínio dos tempos válidos e a cada *chronon* de tempo de validade é associado um subconjunto dos tempos de transação.

Na linguagem de consulta TSQL2, a seleção e a projeção são feitas sobre o rótulo temporal, envolvendo, portanto, tempo de transação e de validade. Essa linguagem de consulta é o resultado da tentativa de vários pesquisadores de definir uma linguagem de consulta temporal padrão [SNO 95]. Seu objetivo foi consolidar diferentes abordagens de modelos de dados temporais, para criar uma linguagem de consulta temporal que, associada a um modelo de dados, venha a ser um consenso entre os pesquisadores, para o desenvolvimento de trabalhos futuros.

Snodgrass [SNO 98] começou a trabalhar com os comitês ANSI (*American National Standards Institute*) e ISO (*International Organization for Standardization*), propondo uma nova parte à SQL3, denominado SQL/Temporal. Isso foi aprovado formalmente em julho de 1995. As discussões começaram então adicionando o suporte a tempo de validade e de transação para SQL3. A essa linguagem foram adicionadas: características de orientação a objetos, as palavras reservadas `validtime`, `transactiontime` e `nonsequenced`, bem como uma semântica temporal formal para a linguagem.

ATSQL é uma linguagem de consulta temporal baseada na SQL [SNO 96]. Suas características principais são os modificadores das declarações (ou *flags*) que podem preceder às consultas para modificar seus comportamentos temporais, controlando a semântica básica das declarações.

Na definição da linguagem de consulta do TVM, denominada TVQL, foi utilizada a SQL como base, adicionando a maioria das características para recuperação temporal das linguagens apresentadas nessa seção. A TVQL é apresentada no próximo capítulo.

2.4.2 Linguagens para Versões

Em uma linguagem de consulta para versões devem existir operações para consultar a estrutura de versões, percorrendo-a através dos relacionamentos entre as versões: derivação, temporal, composição e outros existentes, bem como para obter informações sobre a estrutura como um todo ou sobre cada versão.

As operações que permitem examinar a estrutura de versões podem ser classificadas nos seguintes tipos (com base na taxonomia apresentada em [BJÖ 89]):

- operações que buscam informações sobre uma versão específica (tempo, número ou nome associado a uma versão, bem como o seu estado);
- operações sobre versões relativas a uma versão específica – recuperando com base no tempo, todas as predecessoras ou sucessoras, predecessora ou sucessora imediata. Além disso, recuperando predecessoras ou sucessoras com base no nome (ou caminho), bem como usuários que derivaram a partir desta versão;
- operações sobre o objeto genérico (ou sobre o grafo de versões como um todo) - permitindo a recuperação da primeira, última ou todas as versões, possivelmente restritas pelo nome e/ou tempo. Recuperação da versão *default* ou de configurações;
- operações que buscam um conjunto de versões de um objeto versionado.

Nos últimos anos algumas linguagens de consulta para banco de dados versionados foram propostas: [ABD 97][AND 98][WAG 91].

O ambiente STAR é uma plataforma para o desenvolvimento de aplicações destinadas ao projeto e validação de sistemas digitais [WAG 91]. Esse modelo utiliza conceitos de abstração, classificação, agregação e especialização, para a representação de objetos complexos. Suporta o conceito de versões, representando as diferentes dimensões do processo de evolução do projeto.

Na consulta do ambiente STAR existem duas formas de consultas às informações [MEL 94]. Uma delas é através da linguagem visual de consulta, a qual corresponde à interface de alto nível com o projetista de sistemas digitais. A outra forma de consulta é através da linguagem textual de consulta, a qual oferece ao projetista uma forma alternativa de formulação de consultas, pois possui um poder de expressão maior em relação a uma consulta gráfica.

Com base em SQL, a linguagem textual de consulta é composta pelo bloco `SELECT-FROM-WHERE`, juntamente com a opção `cláusulas-versões` para permitir ao projetista selecionar uma ou mais versões de um objeto [MEL 94]. Essa linguagem de consulta é responsável pela gerência de consultas aos objetos do projeto,

possibilitando a realização de consultas complexas incluindo cláusulas e predicados específicos para o tratamento de versões, bem como dos conceitos de orientação a objetos.

A VQL é uma linguagem de consulta que tem o objetivo de recuperar dados armazenados em um banco de dados multiversionado [ABD 97]. O modelo de dados usado para a definição da VQL é baseado na abordagem de modelo versionado definida por Cellary et al [CEL 90], no qual é possível representar vários estados de um universo modelado.

Do ponto de vista da linguagem, a VQL contribui em dois aspectos:

- especifica termos para denotar o estado do universo modelado – nesse caso, habilita o usuário a especificar quais os estados do universo modelado ele quer consultar;
- realiza operações diferenciadas com relação à dimensão de versão em banco de dados versionados – esse aspecto torna possível manter o caminho de um objeto através do banco de dados versionado (dos diferentes estados do universo modelado).

Nessa linguagem é possível realizar consultas sobre a estrutura de versões e relacionamentos entre as versões, bem como navegar através das versões dos objetos, através de uma seqüência finita de versões de objetos e valores.

A álgebra de consulta proposta por Andonoff é baseada no modelo orientado a objetos proposto por Katz [KAT 90]. Suporta o conceito de tipos, herança de tipos, classes e relacionamentos entre classes [AND 97]. As versões são representadas através de uma hierarquia de derivação como um conjunto de árvores, onde cada árvore corresponde a uma escolha ao projetar a entidade.

Essa álgebra de consulta define um conjunto de operações algébricas correspondentes ao modelo orientado a objetos, tais como: seleção, projeção, junção, groupBy e operações de conjunto. A esse conjunto de operações foi integrado o conceito de versões, onde estas podem ser consultadas com relação à hierarquia de derivação a que pertencem (todos os estados são considerados) ou independentemente umas das outras (somente alguns estados são considerados). Além disso, é possível consultar versões de um tipo que descrevem um tipo versionável [AND 97].

Existe na literatura uma linguagem gráfica de consulta chamada VOHQL (*Version and Object Hypertext Query Language*) que usa a álgebra de Andonoff como base para a definição de sua linguagem [AND 97]. Essa linguagem permite realizar consultas em um banco de dados orientado a objetos integrando o conceito de versões.

Na TVQL, linguagem de consulta do Modelo Temporal de Versões, foram utilizadas características dessas linguagens para a recuperação de versões.

2.5 Considerações Finais

Nesse capítulo foram apresentados alguns modelos de dados e linguagens de consulta com suporte a aspectos temporais e de versões existentes na literatura. Em especial, foi apresentado um modelo que une as características de tempo e de versões chamado TVM. Esse modelo permite a representação de todo o histórico e evolução dos valores dos atributos e relacionamentos das instâncias dos objetos de uma aplicação, bem como as alternativas de projeto.

O próximo capítulo apresenta a TVQL, a linguagem de consulta definida para o modelo TVM, que engloba os aspectos temporais e de versões.

3 TVQL – *Temporal Versioned Query Language*

Depois do modelo TVM definido, o próximo passo é realizar consultas aos dados desse modelo e isso não é possível utilizando-se apenas a SQL padrão, pois esta não manipula dados temporais versionados. Nesse contexto, foi definida uma linguagem de consulta específica para o TVM chamada TVQL – *Temporal Versioned Query Language*. Essa linguagem permite, além da realização de consultas básicas realizadas pela SQL padrão, novas consultas que retornam valores característicos de tempo e versões. A TVQL estabelece um comportamento homogêneo para elementos normais e temporais versionados.

Este capítulo apresenta as características gerais da TVQL e as específicas de tempo e de versões, conforme definido em [MOR 2001b].

3.1 Aspectos Gerais

A TVQL mantém as seguintes características baseadas na SQL: estrutura base (`SELECT-FROM-WHERE`); *alias* para atributos (`SELECT`) e classes (`FROM`); eliminação de valores duplicados (`DISTINCT`); operadores relacionais (`<, >, =, <=, >=, <>`), lógicos (`and, or, not`), de conjuntos (`union, intersection, difference`), de condições compostas (`in, between <valor> and <valor>`) e de combinação de padrões (`like`); funções de agregação (`count, sum, avg, min, max`) e cláusulas para grupo de dados (`group by, having, order by`).

A sintaxe geral da TVQL é apresentada na figura 3.1, onde os colchetes indicam um argumento opcional, as chaves representam argumento repetitivo opcional, os parênteses indicam um conjunto de opções, os argumentos separados pelo caracter | indicam que um deles deve ser utilizado e os argumentos em negrito representam as palavras reservadas da TVQL. No Anexo 1 encontra-se a BNF completa da TVQL.

```
Query ::=     SELECT [ EVER ] [ DISTINCT ]  targetC { , targetC }
              FROM identificC { , identificC }  [ WHERE [ EVER ] searchC ]
              [ GROUP BY groupC { , groupC }  [ HAVING logicalExpr ] ]
              [ ORDER BY orderC { , orderC }  [ setOp query ] ;
targetC ::=   ( * | propertyName | aggregationFunctions | preDefInterval
              | preDefInstant )  [ AS identifier ]
identificC ::=      className [ .VERSIONS ] [ aliasName ]
searchC ::=         logicalExpr | tempExpr
groupC ::=          propertyName | preDefInterval | preDefInstant
logicalExpr ::=     AnyLogicalExpression
tempExpr ::=        logicalExpr | PRESENT ( logicalExpr )
orderC ::=          groupC [ ASC | DESC ]
setOp ::=           UNION | INTERSECTION | DIFFERENCE
preDefInterval ::=  propertyName. ( tInterval | vInterval )
preDefInstant ::=   propertyName. ( tiInstant | tfInstant | viInstant |
                                   vfInstant )
                    | [ className. ] ( iLifeTime | fLifeTime )
```

FIGURA 3.1 – Sintaxe geral da TVQL

3.2 Suporte a Tempo

As consultas que não apresentam nenhuma condição temporal específica nas cláusulas `SELECT` e `WHERE` retornam apenas os valores atuais dos dados dos objetos ativos (não excluídos).

Com relação ao aspecto **temporal**, a TVQL possibilita a realização de consultas sobre o histórico de um objeto, de atributos e relacionamentos desse objeto, além de permitir comparativos entre instantes e intervalos de tempo. Para possibilitar essa funcionalidade, foram definidas várias propriedades para instantes e intervalos, tendo como objetivo a recuperação de rótulos temporais, conforme a tabela 3.1. Vale ressaltar que `iLifeTime` e `fLifeTime` são propriedades predefinidas para objetos e versões, enquanto que as demais propriedades da tabela 3.1 referem-se aos atributos e relacionamentos temporais. Essas propriedades podem ser utilizadas tanto no `SELECT` como no `WHERE` da consulta TVQL.

TABELA 3.1 – Propriedades temporais para instantes e intervalos

Nome	Definição
tInterval	Intervalo de tempo de transação
vInterval	Intervalo de tempo de validade
tiInstant	Tempo de transação inicial
tfInstant	Tempo de transação final
viInstant	Tempo de validade inicial
vfInstant	Tempo de validade final
iLifeTime	Tempo de vida inicial do objeto ou versão
fLifeTime	Tempo de vida final do objeto ou versão

Além das novas propriedades, foi definido também, um conjunto de operadores de comparação específicos para condições temporais. Esses operadores são determinados pelo usuário na cláusula `WHERE` conforme apresentado na tabela 3.2.

Cada um dos operadores de comparação tem uma finalidade específica, que são:

- `INTERSECT` – verifica se um intervalo intersecciona qualquer ponto de outro intervalo;
- `OVERLAP` – verifica se um intervalo intersecciona todo o outro intervalo;
- `EQUAL` – os intervalos são iguais;
- `BEFORE` – um intervalo ou instante antecede totalmente um intervalo;
- `INTO` – um intervalo ou instante está contido em um intervalo;
- `AFTER` – um intervalo ou instante está após um intervalo.

TABELA 3.2 – Operadores de comparação para condições temporais

Condição temporal			Definição
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	Compara os intervalos pré-definidos entre eles
tInterval **vInterval**	**INTERSECT** **OVERLAP** **EQUAL**	**[**`valor1..valor2`**]**	Compara o intervalo de transação ou validade com o intervalo definido entre os instantes `valor1` e `valor2`
		[`..valor`**]**	Compara o intervalo de transação ou validade com o intervalo definido de infinito passado até o instante `valor`
		[`valor..`**]**	Compara o intervalo de transação ou validade com o intervalo definido a partir do instante `valor` até infinito futuro
		TInterval	Compara os intervalos pré-definidos entre eles
tiInstant **tfInstant** **viInstant** **vfInstant** **propriedade** **tInterval** **vInterval**	**BEFORE** **INTO** **AFTER**	**[**`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 o intervalo está antes, dentro ou após o intervalo definido de infinito passado até o instante `valor`
		[`valor..`**]**	Verifica se o instante ou o intervalo está antes, dentro ou após o intervalo definido a partir do instante `valor` até infinito futuro
		tInterval	Verifica se o instante ou o intervalo está antes, dentro ou após o intervalo de transação
		vInterval	Verifica se o instante ou o intervalo está antes, dentro ou após o intervalo de validade

Para representar o uso dessas propriedades e operadores, a consulta abaixo retorna o salário de João Antônio Correia e seu intervalo de validade conforme o que se acreditava em 16/05/2000 (considerando que apenas `salario` é temporal).

```
SELECT EVER salario, salario.vInterval
FROM pessoa
WHERE nome = 'João Antônio Correia'
  AND '16/05/2000' INTO salario.tInterval;
```

Conforme apresentado na consulta anterior, para considerar todos os valores do histórico da vida de objetos e versões, a palavra reservada EVER foi definida. O EVER pode ser usado tanto no SELECT como no WHERE da consulta.

Usando EVER após o SELECT, a consulta retorna o histórico das propriedades selecionadas e considera o histórico das propriedades temporais mencionadas na cláusula WHERE.

Para anular a condição temporal mencionada no comando SQL foi estabelecida a função PRESENT, a qual considera os valores atuais das propriedades. O usuário pode querer consultar propriedades de acordo com valores atuais e históricos ao mesmo tempo.

As consultas da figura 3.2 ilustram as alternativas de consulta com o uso do EVER no SELECT, ou seja, considerando o retorno das propriedades temporais selecionadas. Na figura 3.2a a cláusula WHERE considera também o histórico dos dados, retornando o nome de todas as pessoas que algum dia moraram na rua Onze de maio. Já na figura 3.2b a cláusula WHERE considera apenas os dados atuais, pois a palavra reservada PRESENT anula o EVER anteriormente mencionado no SELECT. Nesse caso, a consulta retorna o histórico dos nomes das pessoas que atualmente moram na rua Onze de maio. Por último a figura 3.2c considera o histórico das propriedades alterando o período de validade (ou transação), retornando o histórico dos nomes das pessoas que moraram na rua Onze de maio, a partir de 01/01/2002.

<table>
<tr><td rowspan="3">SELECT EVER nome
FROM pessoa</td><td>WHERE rua LIKE ‘Onze de maio%’;
(a)</td></tr>
<tr><td>WHERE PRESENT (rua LIKE ‘Onze de maio%’);
(b)</td></tr>
<tr><td>WHERE rua.viInstant >= ‘01/01/2002’;
(c)</td></tr>
</table>

FIGURA 3.2 – Alternativas de consulta com EVER no SELECT

Por outro lado, se o usuário acrescentar o EVER somente na cláusula WHERE, a consulta retorna os valores atuais. No entanto a restrição pode considerar tanto o histórico dos valores, como também os valores atuais da base dependendo de como esta cláusula é apresentada.

Nesse caso, existem também três variações conforme ilustrado na figura 3.3. Na figura 3.3a a cláusula WHERE considera apenas os dados atuais das propriedades, retornando o nome atual das pessoas que atualmente residem na rua Onze de maio. De outro modo, na figura 3.3b a cláusula WHERE, por apresentar o EVER, considera o histórico dos dados, retornando o nome atual das pessoas que algum dia moraram na rua Onze de maio. E, finalmente, a figura 3.3c apresenta uma condição temporal que considera o histórico das propriedades alterando o período de validade (ou transação), retornando o nome atual das pessoas que moraram na rua Onze de maio a partir de 01/01/2002.

SELECT nome FROM pessoa	WHERE rua LIKE 'Onze de maio%'; (a)
	WHERE **EVER** (rua LIKE 'Onze de maio%'); (b)
	WHERE rua.viInstant >= '01/01/2002'; (c)

FIGURA 3.3 – Alternativas de consultas com o **EVER** no **WHERE**

3.3 Suporte a Versões

Para as características de **versionamento**, a linguagem oferece consultas sobre os estados das versões, versão corrente, sobre uma ou várias versões de um objeto, sobre a navegação na hierarquia (ascendentes / descendentes, primeiro / último, predecessores / sucessores, etc) e também sobre as configurações. Considerando a união das características temporais e de versões, as consultas são realizadas sobre as versões, seus estados, configurações e hierarquia, associando isso a um determinado instante ou período de tempo.

Nesse contexto, o usuário define na cláusula `FROM` os objetos e versões a serem consultados. Todas as versões dos objetos são consideradas acrescentando na cláusula `FROM` o nome da classe seguido da palavra reservada `VERSIONS`. Se somente o nome da classe constar na cláusula `FROM`, a consulta considera os objetos e versões correntes.

Por exemplo, considerando a classe `Automóvel` com seus objetos (`Palio`, `Tempra`, `Marea`) e versões (`EX`, `XL`, `EL`, `EM`) conforme apresentados na figura 3.4.

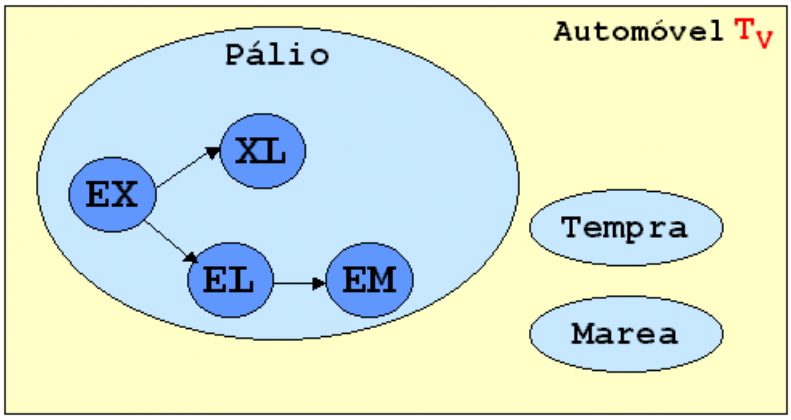

FIGURA 3.4 – Exemplo de classe Temporal Versionada

Considerando as duas cláusulas `FROM` abaixo pertencentes a duas consultas:

- `FROM ClasseX`
- `FROM ClasseX.VERSIONS`

A primeira consulta é realizada sobre os valores dos objetos `Tempra` e `Marea` e a versão corrente do objeto `Palio`. A segunda consulta é realizada sobre os valores dos objetos `Tempra` e `Marea` e sobre as versões do objeto `Palio`: `EX`, `XL`, `EL` e `EM`.

As funções definidas para a recuperação específica das informações de versões e objetos versionados possuem o retorno do tipo `boolean` e podem ser divididas em 3 grupos:

- funções que recuperam o estado (`isWorking`, `isStable`, `isConsolidated` e `isDeactivated`);
- funções da navegação na hierarquia de herança (`isAscendantOf` e `isDescendantOf`);
- funções da navegação na hierarquia de derivação (`isFirst`, `isLast`, `isSuccessorOf`, `isPredecessorOf`, `isCurrent`, `isUserCurrent`, `isConfiguration`).

Para cada função definida para recuperação de estado e para navegação na hierarquia de derivação, existe outra com o mesmo nome e com o sufixo `At`, retornando o valor da informação relativo a um determinado ponto no tempo de validade.

As propriedades para controle do objeto versionado foram definidas para que esses objetos pudessem ser consultados de maneira transparente e podem ser utilizadas tanto no `SELECT` como no `WHERE` do comando TVQL. São elas: `configurationCount`, `currentVersion`, `firstVersion`, `lastVersion`, `nextVersionNumber`, `userCurrentFlag` e `versionCount`.

Duas propriedades são acrescentadas para manipular o apelido dos objetos e entidades armazenadas, `nickname` e `entity` respectivamente.

3.4 Representação de tempo e versões na consulta TVQL

Com o acréscimo das características de tempo e de versões à sintaxe da TVQL, é possível realizar consultas que envolvam, tanto as características de tempo e versões separadamente, como ambos os aspectos na mesma expressão de consulta. No segundo caso, basta utilizar as funções e propriedades definidas na TVQL combinando-as para recuperar ou restringir a consulta em função das características temporais ou de versionamento.

Por exemplo, a consulta a seguir obtém o histórico dos nomes das versões de `Pessoa` cujo estado era consolidado em 18/05/2002.

```
SELECT EVER nome
FROM pessoa.VERSIONS p
WHERE p.isConsolidatedAt('18/05/2002');
```

3.5 Considerações Finais

A TVQL foi definida para contemplar os aspectos temporais e de versionamento do modelo TVM, estabelecendo um comportamento o mais homogêneo possível para elementos normais e temporais versionados.

Esse capítulo apresentou uma visão geral sobre a linguagem de consulta definida para o TVM, englobando as características mantidas da SQL padrão, bem como novas propriedades, comandos e operadores acrescentados para permitir consultas sobre os aspectos temporais e de versionamento.

Com relação ao aspecto temporal, as palavras `EVER` e `PRESENT` foram definidas para recuperação do histórico dos atributos e relacionamentos e para anular a condição temporal mencionada na consulta, respectivamente. Além das palavras reservadas, propriedades temporais e operadores de comparação específicos para instantes e intervalos foram definidos.

Para as consultas referentes às características de versionamento, foi definida a palavra reservada `VERSIONS`, que deve ser utilizada na cláusula `FROM` da consulta TVQL para considerar todas as versões dos objetos. Caso contrário, serão consideradas somente as versões correntes dos objetos. Adicionalmente, algumas funções e propriedades específicas foram definidas para a recuperação das informações de versões e objetos versionados. Em particular, as funções foram divididas em: (i) funções para recuperação do estado; (ii) funções de navegação na hierarquia de herança e; (iii) funções de navegação na hierarquia de derivação.

A especificação completa da TVQL pode ser encontrada na referência [MOR 2001b].

4 Mapeamento do TVM para o DB2

O uso de um modelo temporal eficiente não requer um sistema de gerenciamento de banco de dados específico que suporte a implementação de uma aplicação. Para esse propósito, é possível utilizar um SGDB comercial existente, contanto que seja feito o mapeamento apropriado do modelo temporal versionado para o modelo de dados correspondente ao SGBD adotado. Para mostrar a viabilidade dessa abordagem, as classes do modelo e de aplicação do TVM são mapeadas para um banco de dados convencional relacional.

O objetivo principal desse trabalho é implementar o mapeamento da TVQL para SQL, porém esse mapeamento é realizado considerando uma implementação do TVM sob um banco de dados relacional convencional. Portanto, esse capítulo propõe uma alternativa de implementação do TVM em um banco de dados convencional que será utilizado como base para o mapeamento das consultas. Na figura 4.1, está definido o escopo desse trabalho, ou seja, a implementação do TVM sob um banco de dados relacional convencional (1) e o mapeamento das consultas TVQL nesse banco de dados (2).

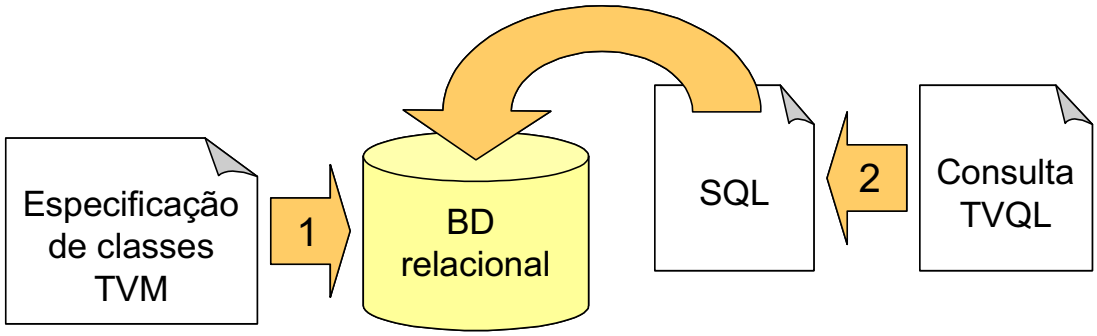

FIGURA 4.1 – Etapas envolvidas no mapeamento

Esse capítulo, inicialmente apresenta os metadados, estrutura base que armazena todas as informações do esquema definido para o TVM. Depois são identificadas três alternativas para o mapeamento dos rótulos temporais das classes, sendo que uma delas é escolhida e detalhada.

4.1 Metadados

Os metadados armazenam as informações específicas do esquema da base modelada com o TVM. Esse trabalho não aborda o mapeamento dos metadados, pois considera que estes já estão criados no banco de dados relacional. No entanto, é utilizada a estrutura proposta em [MOR 2001b], conforme ilustrado na figura 4.2.

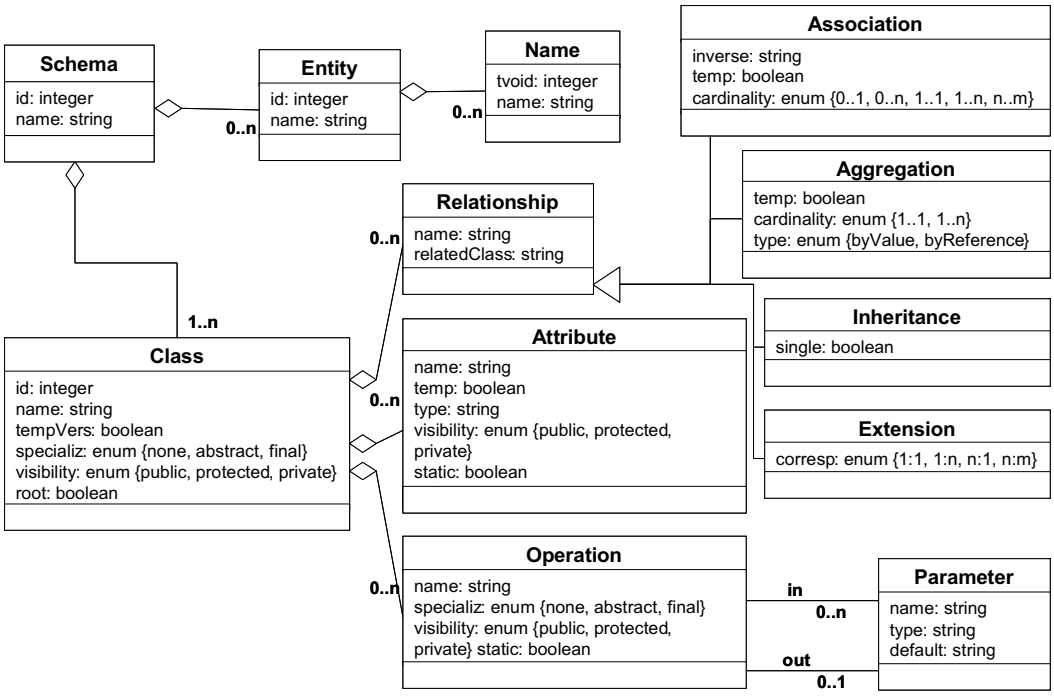

FIGURA 4.2 – Estrutura dos metadados

Nessa estrutura, a classe `Entity` contém atributos para identificador e nome da entidade. A classe `Class` possui atributos para identificador e nome da classe, bem como para gerenciar se a classe é temporal versionada. Possui o atributo `especializ` que determina o tipo de especialização que pode ser realizada a partir da classe. Possui também um atributo para determinar a visibilidade da classe e para indicar se é uma raiz de uma hierarquia de herança. A classe `name` possui atributos para identificador e nome de objetos. A classe abstrata `Relationship` possui atributos para seu nome e nome da classe relacionada e possui quatro especializações: `Association`, `Aggregation`, `Inheritance` e `Extension`. A classe `Attribute` possui atributos para nome, para indicar se é atributo temporal, para visibilidade, para determinar se é estático e valor *default*. A classe `Operation` possui atributos para nome da operação, tipo de especialização, visibilidade e se é estática. A classe `Parameter` possui atributos para nome, tipo e valor *default*.

4.2 Mapeamento das Classes do Modelo

Para implementar o mapeamento foi utilizado um BD relacional (DB2), o que leva à necessidade de mapear as classes do modelo TVM para esse BD. Sendo assim, alguns passos devem ser seguidos para mapear o TVM, conforme descritos nesta seção.

A figura 4.3 apresenta a hierarquia de classes do TVM juntamente com seus respectivos atributos, os quais são mapeados para o DB2. O atributo `tvoid` da classe `Object` é comum a todas as classes da aplicação, com a diferença que para as classes sem tempo e versões, o número da versão não possui valor. O atributo `alive` da classe `TemporalObject` armazena o estado de vida do objeto (`true` no momento da criação e `false` quando o objeto for excluído logicamente). Esse atributo é temporal, pois os objetos temporais podem ser restaurados.

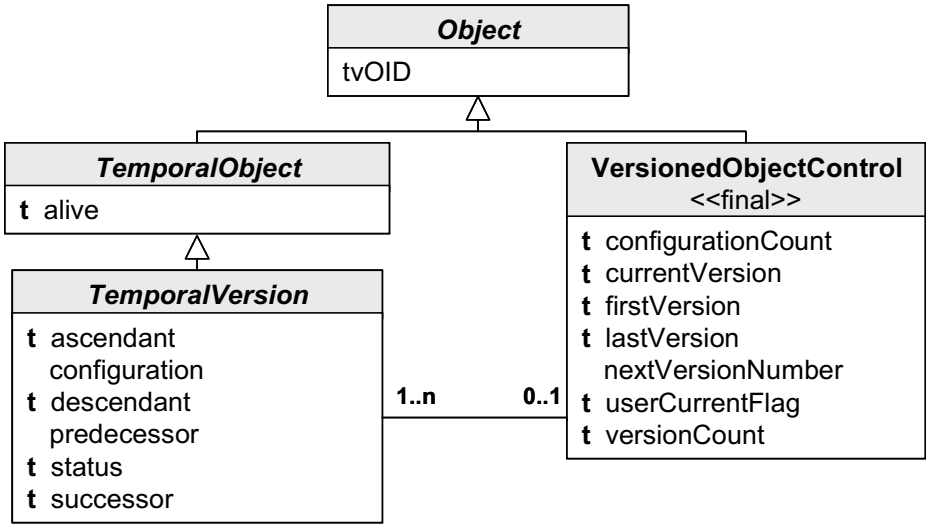

FIGURA 4.3 – Hierarquia de classes TVM com atributos

Na classe `VersionedObjectControl` estão as propriedades para controle dos objetos versionados, sendo que os atributos `configurationCount`, `currentVersion`, `firstVersion`, `lastVersion`, `userCurrentFlag` e `versionCount` são definidos como temporais, pois seus valores podem evoluir com o tempo e o histórico deve ficar armazenado. Os valores são inseridos e atualizados nessa tabela a cada derivação (`firstVersion`, `lastVersion`, etc).

Partindo da hierarquia base do TVM mostrada na figura 4.3, a única que necessita mapeamento direto é a `VersionedObjectControl`. Essa classe é mapeada para uma tabela denominada `VOC`, tendo como chave primária o atributo `tvoid`, conforme a estrutura ilustrada na figura 4.4. Esta tabela armazena informações que permitem controlar todas as versões e objetos versionados.

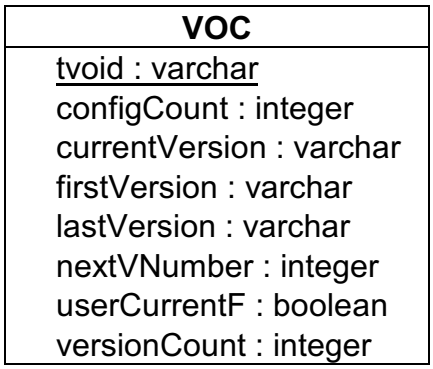

FIGURA 4.4 – Estrutura da tabela `VOC`

As classes `TemporalObject` e `TemporalVersion` ficam implícitas quando mapeadas para o banco de dados relacional. Os atributos dessas classes são inseridos nas classes de aplicação no momento da especificação para o banco de dados relacional (BDR).

O identificador dos objetos do TVM (`tvoid`), definido na classe `Object`, continua sendo composto pela classe, entidade e versão. Esse identificador é gerado automaticamente no momento da instanciação através de *triggers*. Ele é do tipo

varchar de tamanho 20, sendo que a cada 6 caracteres são armazenados seqüencialmente os valores da classe, entidade versão separados por vírgula (figura 4.5).

FIGURA 4.5 – Estrutura do tvoid

Como o DB2 não possui tipo de conjunto, os atributos temporais ascendant, descendant, predecessor e successor necessitam mapeamento especial. Para evitar redundâncias, esses atributos são mapeados para duas tabelas auxiliares, conforme mostrado na figura 4.6.

PredSucc
seqPS
Predecessor : varchar
Successor : varchar
itTime : timestamp
ftTime : timestamp
ivTime : timestamp
fvTime : timestamp

AscDesc
seqAD
Ascendant : varchar
Descendant : varchar
itTime : timestamp
ftTime : timestamp
ivTime : timestamp
fvTime : timestamp

FIGURA 4.6 – Mapeamento dos atributos temporais (PredSucc e AscDesc)

Essas tabelas armazenam os pares predecessor-sucessor e ascendente-descendente de todas as classes especificadas, com os respectivos rótulos temporais. A chave primária é composta por um seqüencial, pois os pares podem possuir valores nulos.

As tabelas PredSucc e AscDesc são mantidas pelo sistema. Somente na instanciação e na derivação são armazenados os predecessores e sucessores bem como os ascendentes e descendentes. Depois de inseridos na tabela, esses valores só podem ser alterados no momento da exclusão da versão, encerrando os tempos de validade dos pares nos quais a versão excluída está presente. Essas tabelas só podem ser alteradas pelo sistema, ou seja, pelos gatilhos (*triggers*) e procedimentos armazenados (*stored procedures*) criados no gerenciamento.

No Anexo 2 podem ser encontrados os comandos SQL necessários para a criação das tabelas resultantes do mapeamento das classes da hierarquia do TVM.

Os demais atributos temporais da tabela VOC (VersionedObjectControl) seguem as regras de mapeamento especificadas para mapear as classes de aplicação, e que são discutidas nas próximas seções.

4.3 Formas para o Mapeamento das Classes de Aplicação

Depois de realizado o mapeamento das classes do modelo, a próxima etapa envolve o mapeamento das classes de aplicação. Nesse caso, para manter as

características temporais dos atributos e relacionamentos, foram identificadas três maneiras para mapear as classes: (i) armazenando os rótulos por tuplas; (ii) gerando todo o histórico em uma tabela; ou (iii) criando uma tabela auxiliar para representar cada um dos atributos e relacionamentos temporais.

Para exemplificar cada uma das alternativas encontradas considera-se a classe `Pessoa`, com os atributos `nome` do tipo `string`, `cpf integer` e `salario real`, onde `nome` e `salario` são temporais, conforme a figura 4.7.

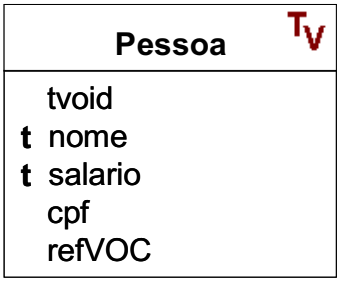

FIGURA 4.7 – Modelo de classe para mapeamento

Além disso, para cada classe definida no modelo TVM existe uma tabela correspondente no banco de dados relacional. Esta tabela é chamada de tabela principal.

4.3.1 Armazenamento dos rótulos temporais por tuplas

Nesta alternativa a idéia é deixar os rótulos temporais dos atributos na tabela à qual pertencem. Assim, para cada tupla é criado um conjunto de rótulos temporais para registrar seus tempos de validade e transação (inicial e final). Todos esses atributos são criados na tabela principal. Na figura 4.8 é mostrada a classe `Pessoa` com seus atributos temporais `nome` e `salario` que aparecem juntamente com o rótulo temporal da tupla.

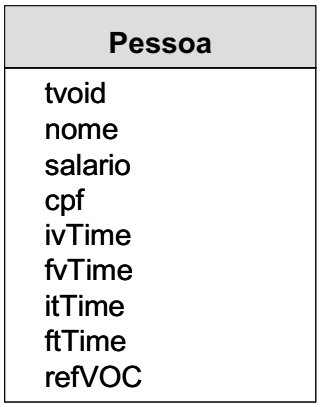

FIGURA 4.8 – Mapeamento dos rótulos temporais por tuplas

Essa alternativa tem a desvantagem de não informar diretamente os tempos de uma única coluna. Os rótulos temporais são associados à tupla inteira. Como a maior parte das consultas é realizada sobre os valores atuais, essa alternativa pode comprometer o desempenho no caso das consultas atuais, por aumentar significativamente o volume de informação armazenada na tabela mapeada com a redundância dos valores que permanecem inalterados nas tuplas.

4.3.2 Armazenamento dos rótulos temporais em somente uma tabela auxi liar

Outra possibilidade é criar uma tabela auxiliar onde todos os dados históricos ficariam armazenados, independentemente da classe à qual pertencem e do tipo da propriedade. A figura 4.9 ilustra esta forma de mapeamento, através da classe `Pessoa` e suas propriedades. A tabela `Temporal` é composta pelo `tvoid`, o nome do atributo temporal (`nomeAtributo`), o valor do atributo e os rótulos temporais, e é responsável por armazenar todos os valores de todos os atributos temporais da aplicação.

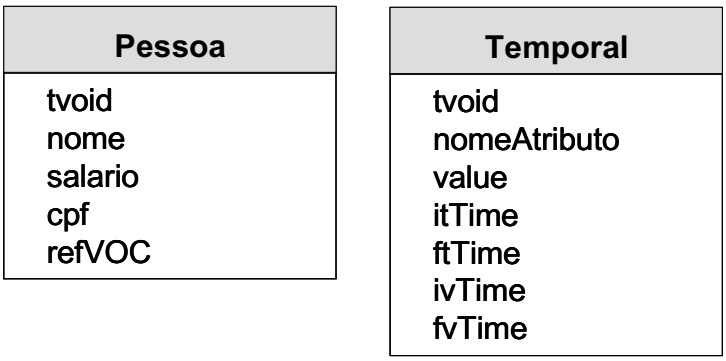

FIGURA 4.9 – Mapeamento com rótulos temporais em uma só tabela

Entretanto essa alternativa também é inviável, principalmente porque o domínio dos tipos dos atributos varia. Exemplificando melhor, se considerada a figura 4.9, o histórico dos atributos `nome` e `salario` estariam armazenados na tabela `Temporal`, mesmo sabendo que `nome` é do tipo `string` e `salario` é `real`. Assim o atributo `value` da tabela `Temporal` teria que ser de um tipo genérico o bastante para suportar qualquer um dos tipos de atributos existentes no DB2.

Além disso, esta alternativa poderia comprometer o desempenho quando a consulta envolvesse informações históricas do banco de dados devido ao volume de informações armazenadas na tabela `Temporal`, uma vez que o histórico de toda a aplicação estaria armazenado nesta tabela.

4.3.3 Criação de tabelas auxiliares para armazenar os atributos temporais

Essa forma de mapeamento consiste também na criação de uma tabela para cada classe da aplicação, no entanto é 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 (figura 4.10).

Dessa forma, a tabela principal possui colunas que representam os atributos (temporais ou não temporais), além dos atributos necessários para o controle das versões. Como para cada atributo temporal existe uma tabela auxiliar relacionada, o problema de domínio de valores, apresentado na alternativa anterior, está resolvido.

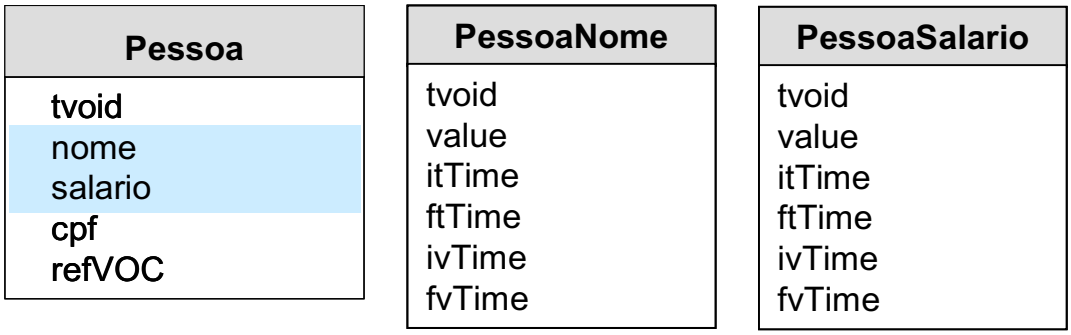

FIGURA 4.10 – Rótulo temporal representado por tabelas auxiliares

As informações atuais são mantidas na tabela principal. No entanto, foram analisadas duas formas para armazenar os valores dos atributos temporais usando essa abordagem: (i) mantendo todos os valores dos atributos temporais na tabela auxiliar, inclusive o valor atual; e (ii) mantendo somente o histórico na tabela auxiliar. A primeira é mais completa, pois permite o armazenamento dos rótulos temporais das informações atuais (o que não seria possível na segunda). Além disso, na primeira forma, para uma consulta a um atributo temporal em um ponto no passado até hoje, por exemplo, basta retornar os valores da tabela auxiliar, enquanto que no segundo caso, seria necessário fazer uma união das informações das tabelas principal e auxiliar para conseguir o resultado esperado.

Portanto, nessa alternativa as informações atuais dos atributos temporais ficam armazenadas tanto na tabela principal como na auxiliar correspondente, pelas seguintes razões:

- o valor atual na tabela principal agiliza a consulta a informações atuais, uma vez que a maioria das consultas é feita sobre esses valores;
- o valor atual na tabela auxiliar mantém os rótulos da informação atual, ou seja, os tempos de transação e validade inicial e o tempo de validade final, caso seja atribuída uma validade futura. Além disso, facilita a recuperação de valores históricos e atuais na mesma consulta.

A terceira alternativa foi a que apresentou melhores condições para gerenciar os aspectos temporais, sem grandes prejuízos às consultas nem à estrutura das tabelas principais. Por esse motivo, o presente trabalho propõe essa alternativa como solução para o mapeamento do TVM e a próxima seção apresenta o seu detalhamento.

4.4 Mapeamento das Classes de Aplicação

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

- cada atributo é mapeado para uma coluna na tabela com os respectivos nome e tipo;
- os tipos de dados são mapeados conforme a tabela 4.1;

TABELA 4.1 – 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
Oidt	Varchar (20)
Set	-

- enumerações são mapeadas para regras de integridade (*constraints*) que contêm os valores especificados relacionadas à respectiva coluna;
- todas as classes recebem o atributo `tvoid` do tipo `varchar` como chave primária.

As classes temporais versionadas são mapeadas também para tabelas principais como as tabelas normais, no entanto recebem por *default* os atributos herdados 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, conforme apresentado na seção 4.2;
- toda tabela principal possui uma referência (chave estrangeira) chamada `refVOC`, que referencia o identificador (`tvoid`) da tabela de controle de objetos versionados (tabela `VOC`).

Exemplificando as etapas do mapeamento, a figura 4.11 ilustra a estrutura do mapeamento de uma classe de aplicação temporal versionada, conforme descrito anteriormente.

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.

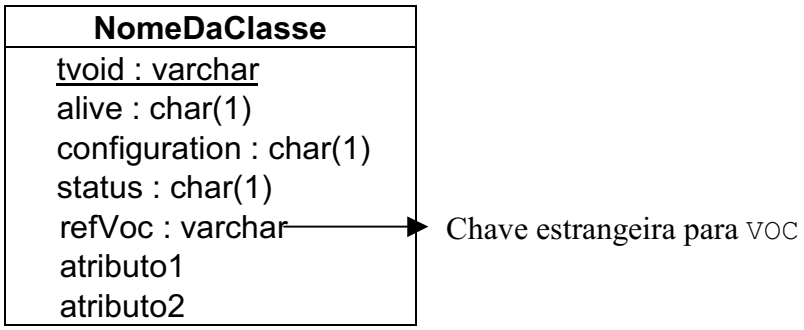

FIGURA 4.11 – Mapeamento de Classes de aplicação

Na herança simples as subclasses herdam os atributos da superclasse, ou seja, todos os atributos e relacionamentos da superclasse são adicionados na subclasse. Se a superclasse for abstrata, ela não é instanciada e, portanto, não gera tabela no esquema relacional. Por outro lado, se a superclasse não for abstrata, o seu mapeamento segue as regras definidas para as classes, conforme descritos anteriormente.

Os atributos e relacionamentos temporais, definidos nas classes temporais versionadas, são mapeados em tabelas específicas denominadas tabelas auxiliares, as quais armazenam seus históricos. Essas tabelas, quando mapeadas, apresentam as características descritas a seguir e ilustradas na figura 4.12:

- 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 `itTime`, `ftTime`, `ivTime`, `fvTime` 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);
- a chave primária da tabela auxiliar é composta pelo identificador (`tvoid`) e os tempos iniciais de transação (`itTime`) e validade (`ivTime`).

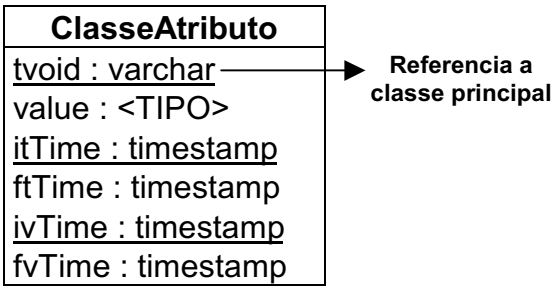

FIGURA 4.12 – Mapeamento das tabelas auxiliares

As tabelas principais devem possuir gatilhos (*triggers*) associados que controlam os rótulos temporais. Essa parte é deixada para o mapeamento do gerenciamento do modelo, não sendo abordada nesse trabalho.

Um *script* exemplificando o processo de criação de tabelas referentes a classes de uma aplicação pode ser encontrada no Anexo 3.

Nas tabelas principais ficam armazenados somente os valores atuais dos atributos temporais, pois a maior parte das consultas é realizada sobre os dados atuais do BD. No entanto nas tabelas auxiliares são armazenados tanto os valores históricos como os valores atuais dos atributos. Esse tipo de implementação facilita a consulta, pois somente as tabelas principais são consideradas caso a consulta seja atual. Da mesma forma, se a consulta for temporal somente os valores armazenados nas tabelas auxiliares são retornadas na consulta.

A cada nova tabela criada no banco de dados, são inseridas as informações destas tabelas nos metadados. São armazenadas as informações sobre a entidade, classe, atributo e relacionamento.

4.5 Considerações Finais

Esse capítulo apresentou as regras para o mapeamento das classes do modelo e de aplicação do TVM para o DB2. Das classes da hierarquia do TVM, a classe `VersionedObjectControl` é a única que necessita de mapeamento direto, sendo 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 tipo de conjunto. Para cada atributo temporal definido na tabela `VOC`, existe uma tabela auxiliar correspondente. A figura 4.13 representa o mapeamento das classes da hierarquia do TVM, conforme exposto nesse capítulo.

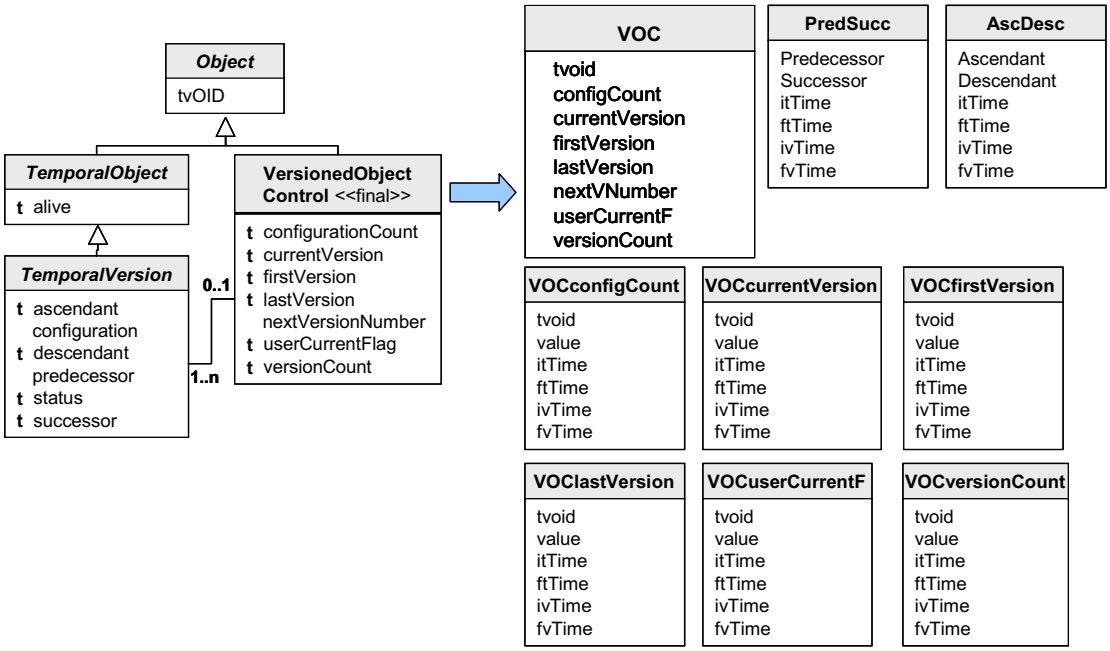

FIGURA 4.13 – Mapeamento da hierarquia de classes TVM para o DB2

Quanto às classes de aplicação, em resumo, elas são mapeadas para tabelas e os atributos para colunas nessas tabelas. Para os atributos temporais são criadas tabelas auxiliares que armazenam o histórico dos dados destes atributos. As tabelas auxiliares possuem como colunas o valor do atributo temporal e os rótulos temporais, bem como o `tvoid` da tabela principal. As tabelas principais possuem como colunas, o `tvoid`, os

atributos de `TemporalObject` (`Alive`) e `TemporalVersion` (`Configuration` e `Status`), além dos atributos definidos na classe que está sendo mapeada.

Na figura 4.14 é mostrado o mapeamento completo da classe `Pessoa`, com os atributos `nome`, `rua`, `salario` e `cpf` sendo `nome` e `salario` temporais. O mapeamento resulta em uma tabela chamada `Pessoa` com as colunas equivalentes aos atributos da classe mais os atributos provenientes das Classes `TemporalObject` e `TemporalVersion`, pois estas ficam implícitas quando mapeadas para o DB2. Além disso, a tabela `Pessoa`, possui uma referência (`refVOC`) à tabela `VOC` que corresponde à tabela de controle de objetos versionados. Para cada um dos atributos temporais é criada uma tabela que armazena o histórico do atributo juntamente com seus rótulos temporais.

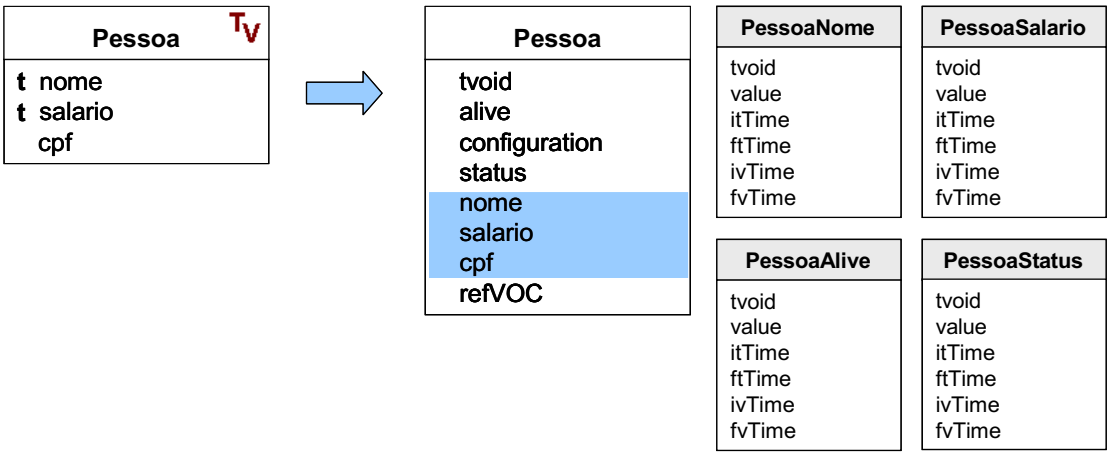

FIGURA 4.14 – Exemplo do mapeamento de classes de aplicação para o DB2

5 Mapeamento da TVQL para a SQL-92

Depois de realizar o mapeamento das classes do modelo TVM e das classes de aplicação, este capítulo tem o objetivo de implementar o mapeamento das funções e propriedades definidas na TVQL para a SQL. São descritas duas alternativas identificadas para o mapeamento da TVQL e finalmente, a partir da escolha de uma das alternativas, é apresentado o algoritmo completo para o mapeamento.

5.1 Introdução ao mapeamento da TVQL

O mapeamento da TVQL para SQL depende da estrutura gerada pelo mapeamento da especificação de classes para o banco de dados suporte, conforme apresentado no capítulo 4.

Considerando que todas as classes da especificação estão mapeadas para tabelas relacionais no DB2, chamadas **tabelas principais**, atributos e relacionamentos temporais também estão mapeados para tabelas próprias, chamadas **tabelas auxiliares**, com um relacionamento para a tabela principal. O controle dos objetos versionados é armazenado em uma única tabela, chamada `VOC`, com a qual todas as tabelas com mais de uma versão possuem relacionamento. Partindo desse pressuposto, o processo de mapeamento da TVQL pode ser iniciado.

Em princípio, as características gerais da TVQL baseadas na SQL permanecem iguais no mapeamento. Ou seja, os *aliases*, a eliminação de duplicatas (`DISTINCT`), os operadores, as funções de agregação (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`) e as cláusulas de grupo de dados (`GROUP BY`, `HAVING`, `ORDER BY`) não são alterados na consulta SQL. As classes e propriedades envolvidas nessas operações, funções e cláusulas são mapeados nas próximas seções.

5.2 Alternativas para o Mapeamento

Durante a elaboração desse trabalho foram identificadas duas formas para realizar o mapeamento, que se diferenciam pela forma como as cláusulas TVQL são avaliadas e pelo critério de construção do comando SQL. Essas alternativas estão relatadas a seguir e, posteriormente, a alternativa escolhida é detalhada.

5.2.1 Mapeamento das cláusulas TVQL

Na primeira alternativa parte-se da leitura da cláusula `FROM` da TVQL e monta as cláusulas `FROM` e `WHERE` da SQL. Depois, lendo a cláusula `SELECT` da TVQL, as cláusulas `SELECT`, `FROM` e `WHERE` são montadas. Por último, é realizada a leitura da cláusula `WHERE` da TVQL e são montadas as cláusulas `FROM` e `WHERE` da SQL, conforme apresentado na tabela 5.1.

TABELA 5.1 – Primeira alternativa para o mapeamento da TVQL

Alternativa A	
Lê TVQL	**Constrói SQL**
FROM	FROM WHERE
SELECT	SELECT FROM WHERE
WHERE	FROM WHERE

Considerando a classe `Pessoa`, a figura 5.1 ilustra a forma como é realizado o mapeamento da TVQL para SQL, conforme definido nessa alternativa.

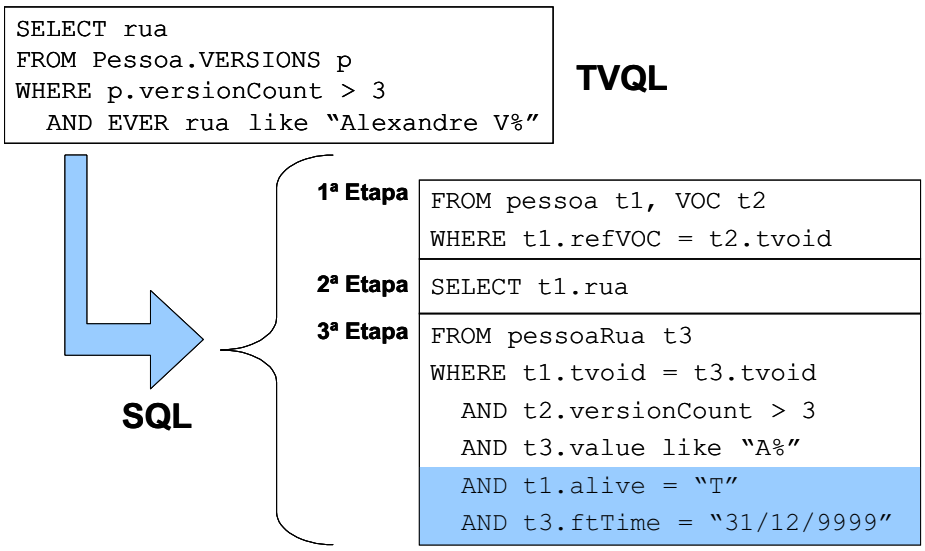

FIGURA 5.1 – Primeira alternativa para o mapeamento da TVQL

Usando essa proposta, no momento de mapear a cláusula `WHERE` não é possível fazer sua construção completa, pois algumas restrições especiais são necessárias e não estão expressas na cláusula `WHERE` da TVQL. Para determinar quais condições especiais são inseridas na cláusula `WHERE` da SQL, é preciso analisar todo o comando TVQL. A partir dessa constatação, optou-se por usar a mesma regra para mapear todas as cláusulas da TVQL, ou seja, percorrer toda a TVQL e ir construindo cada cláusula da SQL. Essa proposta é apresentada na próxima seção.

5.2.2 Construção das cláusulas SQL a partir da leitura da TVQL

Esta forma de mapeamento da TVQL consiste em construir as cláusulas da SQL a partir da leitura realizada na TVQL. A construção da SQL começa também pela cláusula `FROM`, depois a cláusula `SELECT` e por último as restrições da consulta (cláusula `WHERE`).

Nesse processo, para construir a cláusula `FROM` da SQL, é necessário percorrer toda a consulta TVQL para verificar quais tabelas devem fazer parte da consulta mapeada, podendo ser tabelas principais, tabelas auxiliares ou ainda a tabela `VOC`. Para a cláusula `SELECT` apenas as cláusulas `SELECT` e `FROM` da TVQL são avaliadas, pois é

necessário identificar o atributo a ser retornado, se o seu aspecto temporal é considerado ou não, e a qual tabela pertence. Na construção da cláusula `WHERE`, todas as cláusulas da TVQL são novamente avaliadas, pois nessa cláusula estão as maiores verificações e traduções a serem feitas, desde condições especiais a serem acrescentadas até o mapeamento das propriedades e funções tanto temporais como de versionamento, especificadas na restrição da consulta.

A tabela 5.2 mostra as cláusulas da TVQL que são verificadas para montar cada uma das cláusulas da SQL.

TABELA 5.2 – Segunda alternativa para o mapeamento da TVQL

Alternativa B	
Lê TVQL	**Constrói SQL**
SELECT FROM WHERE	FROM
SELECT FROM	SELECT
SELECT FROM WHERE	WHERE

Na figura 5.2 é apresentado um exemplo de mapeamento de uma consulta TVQL para SQL, segundo essa abordagem de implementação. É utilizada a mesma consulta TVQL expressa na alternativa de mapeamento descrita na seção anterior.

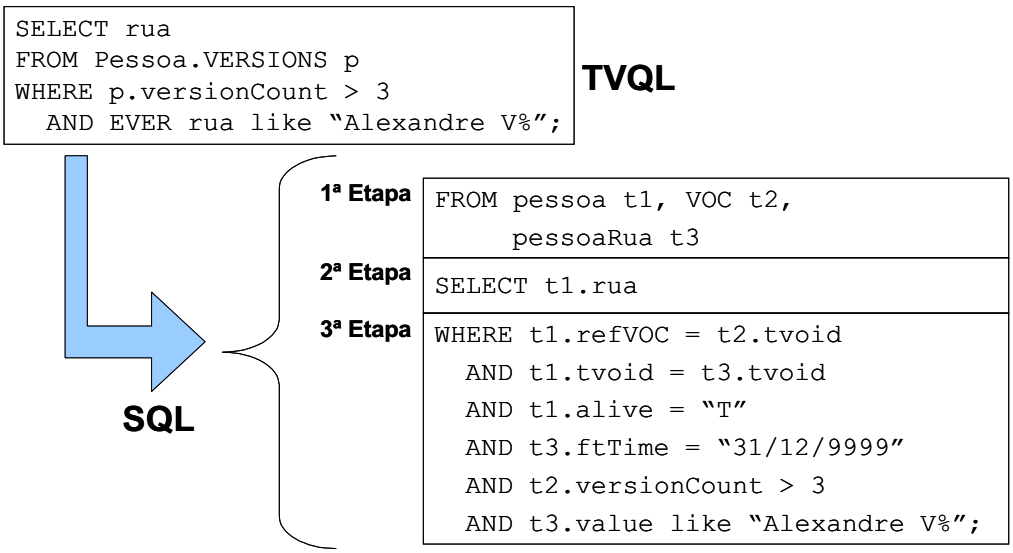

FIGURA 5.2 – Segunda alternativa para o mapeamento da TVQL

A grande vantagem dessa alternativa é que cada cláusula SQL é construída em sua totalidade, sem a necessidade de alterações posteriores. Além disso, a mesma regra é utilizada para construir todas as cláusulas da consulta. Por esses motivos, a proposta atual é utilizada para implementar o mapeamento da TVQL.

5.3 Detalhamento da Alternativa Escolhida

Considerando a idéia da segunda alternativa, esta seção detalha o mapeamento da TVQL para SQL começando pela cláusula `FROM`, seguindo pelas cláusulas `SELECT` e `WHERE`, dividido em quatro passos: (i) mapeamento de classe, atributos e associações; (ii) inclusão de condições especiais; (iii) mapeamento de restrições temporais; e (iv) mapeamento das restrições de versionamento.

5.3.1 Classes, Atributos e Associações

O objetivo dessa etapa é identificar em que tabelas do banco de dados as classes e as propriedades selecionadas estão armazenadas e como se relacionam. O mapeamento começa pela geração da cláusula `FROM` do SQL, percorrendo a consulta TVQL toda para identificar quais tabelas participam da consulta. Nessa cláusula, as classes especificadas são mapeadas para as tabelas correspondentes, que recebem um *alias* (`t1,t2,...tn`). Na cláusula `SELECT`, os atributos são mapeados para as respectivas colunas das tabelas definidas no `FROM`.

Antes de realizar o mapeamento de qualquer consulta TVQL, é feita a verificação se as tabelas envolvidas são temporais versionadas. As consultas que não apresentam classes temporais versionadas não necessitam de mapeamento específico. Para as consultas às classes que apresentam tempo e versões, várias possibilidades devem ser consideradas. Essas possibilidades são agrupadas em casos e apresentadas a seguir.

Gerando a Cláusula `FROM` do SQL

Para construir a cláusula `FROM` do SQL, é necessário percorrer todo o comando TVQL para identificar com precisão quais tabelas participam da consulta. Para cada propriedade mencionada nas cláusulas `SELECT` e `WHERE` e para cada classe mencionada na cláusula `FROM`, é feita uma consulta nos metadados para verificar em que tabela está armazenada. Ao final desse processo, a cláusula `FROM` estará completa, não sendo necessário alterá-la novamente.

O mapeamento das cláusulas `SELECT` e `WHERE` não está sendo realizado neste momento. O mapeamento de cada cláusula será mostrado a medida que as etapas vão sendo cumpridas.

Caso 1. (cláusula `FROM` e a palavra reservada `VERSIONS`). A tabela `VOC` é a tabela responsável pelo controle dos objetos versionados do modelo TVM. A inclusão dessa tabela no mapeamento é feita em duas ocasiões:

- quando a consulta é realizada sobre a versão corrente; ou
- quando a consulta é feita sobre várias versões ou objetos versionados, mas retornando ou filtrando alguma característica que envolva os aspectos de versionamento.

No caso de classes temporais versionadas, quando a palavra reservada `VERSIONS` não está especificada na consulta, significa que são considerados somente objetos e versões correntes. Então, a tabela `VOC` é incluída na cláusula `FROM` do SQL, conforme mostrado a seguir (considerando `Pessoa` como uma classe temporal versionada e `nome` um atributo temporal):

TVQL	SQL (FROM)
SELECT nome FROM Pessoa;	SELECT ... FROM **pessoa t1, VOC t2** WHERE ...

A tabela VOC também é necessária quando a consulta apresenta a palavra reservada VERSIONS juntamente com alguma das funções de navegação na hierarquia de derivação ou alguma das seguintes propriedades: configurationCount, currentVersion, firstVersion, lastVersion, nextVersionNumber ou userCurrentFlag.

Nesses casos, faz-se necessário o uso da tabela VOC, pois ela contém os atributos que gerenciam os objetos versionados. Alterando a consulta anterior, se o objetivo fosse retornar o nome e o tvoid da primeira versão de todas as versões de pessoa, a consulta seria escrita e a cláusula FROM mapeada como segue:

TVQL	SQL (FROM)
SELECT nome, firstVersion FROM Pessoa.VERSIONS;	SELECT ... FROM **pessoa t1, VOC t2** WHERE ...

A critério de simplificação, os exemplos de consultas TVQL utilizados na especificação das cláusulas FROM e SELECT da SQL serão apresentados com a palavra reservada VERSIONS, pois maiores detalhamentos e condições especiais são necessárias no momento da construção da cláusula WHERE.

Caso 2. (cláusula FROM sem a palavra reservada EVER na consulta). Em consultas que não apresentem aspectos temporais, o mapeamento da cláusula FROM considera apenas as tabelas principais, ou seja, as tabelas mencionadas no comando TVQL. Por exemplo, sendo nome um atributo temporal da classe temporal versionada Pessoa, considere a consulta na linguagem TVQL (à esquerda) que recupera os nomes atuais de todas as versões pessoas. À direita é possível observar que o mapeamento da cláusula FROM para SQL é simples e direto:

TVQL	SQL (FROM)
SELECT nome FROM Pessoa.VERSIONS;	SELECT ... FROM **pessoa t1** WHERE ...

A cláusula FROM na sintaxe SQL anterior, não utiliza a tabela VOC, pois a consulta refere-se a todas as versões de Pessoa. Essa informação é obtida apenas com o uso da tabela principal.

Caso 3. (FROM com a palavra reservada EVER e atributos temporais). Se a cláusula SELECT apresenta a palavra reservada EVER e existem atributos temporais envolvidos na consulta, as tabelas auxiliares desses atributos devem ser inseridas na cláusula FROM do SQL.

Considerando a classe Pessoa, a cláusula FROM da consulta TVQL é mapeada conforme apresentado a seguir:

TVQL	SQL (FROM)
SELECT EVER nome FROM Pessoa.VERSIONS;	SELECT ... FROM pessoa t1, pessoaNome t2 WHERE ...

Em todas as consultas TVQL, quando a cláusula `FROM` da SQL estiver sendo construída, a tabela principal deve estar presente. Os principais motivos são que:

- nas consultas a todas as versões, por *default*, o retorno corresponde a todas as versões ativas. O atributo que representa o estado de "vida" de um objeto ou versão está na tabela principal (atributo `alive`);
- nas consultas a todas as versões, mesmo que o usuário não queira recuperar as versões ativas, ele terá que especificar na consulta o estado das versões que deseja retornar. Mais uma vez é preciso acessar o atributo `status` na tabela principal;
- nas consultas às versões correntes a tabela `VOC` participa da cláusula `FROM`, o que obriga o uso da tabela principal, pois esta é a única que possui referência (chave estrangeira) à tabela `VOC`.

Caso 4. (`FROM` com a palavra reservada `EVER`, considerando atributos temporais e não temporais). Quando na consulta TVQL constarem atributos temporais e atributos não temporais, a cláusula `FROM` da SQL terá necessariamente as tabelas auxiliares e as principais. Por exemplo, considerando a consulta anterior, mas acrescentando no `SELECT` o atributo `cpf` (não temporal) da `Pessoa`, a cláusula FROM do comando SQL é construída com a adição das tabelas principal e auxiliar:

TVQL	SQL (FROM)
SELECT EVER nome, cpf FROM Pessoa.VERSIONS;	SELECT ... **FROM pessoa t1, pessoaNome t2** WHERE ...

De outro modo, quando a consulta TVQL apresenta a função `PRESENT` em sua construção, são considerados os valores atuais das propriedades temporais mencionadas dentro do escopo da função. Exemplificando essa definição, a próxima consulta retorna o histórico do nome de todas as versões das pessoas que atualmente recebem salário superior a 2000 (considerando que nome e salário são atributos temporais):

TVQL	SQL (FROM)
SELECT EVER nome, cpf FROM Pessoa.VERSIONS WHERE PRESENT(salario> 2000);	SELECT ... **FROM pessoa t1, pessoaNome t2** WHERE ...

Apesar de todos os atributos envolvidos serem temporais, a palavra reservada `PRESENT` faz com que o valor atual do atributo seja considerado. Por essa razão, as duas tabelas participam da consulta mapeada.

Caso 5. (`FROM` sem a palavra reservada `EVER`, mas com as propriedades temporais, operadores de comparação temporal ou funções com sufixo `At`). Existe ainda a possibilidade de a consulta TVQL não possuir a palavra reservada `EVER`, mas algumas das propriedades temporais estejam presentes (`tiInstant`, `tfInstant`, `viInstant`, `vfInstant`, `tInterval`, `vInterval`, `iLifeTime`, `fLifeTime`). Além das

propriedades temporais, podem estar especificadas as funções para controle de objetos versionados que possuem o sufixo `At`.

Em todos esses casos, na cláusula `FROM` da SQL devem participar as tabelas as auxiliares dos atributos relacionados. Por exemplo, para saber o instante de todas as alterações do `nome`, nas versões de `Pessoa` no estado atual da base. A consulta TVQL e o mapeamento da cláusula `FROM` são os seguintes:

TVQL	**SQL (FROM)**

`SELECT DISTINCT nome.tiInstant`
`FROM Pessoa.VERSIONS;`
	`SELECT ...`
	`FROM` **`pessoa t1, pessoaNome t2`**
	`WHERE ...`

Com funções que possuem o sufixo At, a consulta e o mapeamento são representados da seguinte forma:

TVQL	**SQL (FROM)**

`SELECT nome`
`FROM Pessoa.VERSIONS p`
`WHERE p.isFirtAt('18/05/2002');`
	`SELECT ...`
	`FROM` **`pessoa t1, VOC t2,`**
	`VOCfirstVersion t3`
	`WHERE ...`

Essa consulta recupera o nome das versões de pessoa que eram primeiras versões em 18/05/2002. O atributo `firstVersion` gerencia essa informação e é um atributo temporal, portanto a função `isFirstAt` busca nos rótulos temporais do atributo `firstVersion`, as informações que eram válidas naquela data.

Caso 6. (funções de navegação na hierarquia de herança e funções `isSuccessorOf`, `isSuccessorOfAt` e `isPredecessorOf`). Para as consultas TVQL que apresentam as funções `isSucessorOf`, `isSucessorOfAt` e `isPredecessorOf`, o mapeamento da cláusula `FROM` da SQL deve inserir a tabela `PredSucc`. Por exemplo:

TVQL	**SQL (FROM)**

`SELECT p1.nome, p2.tvoid`
`FROM Pessoa.VERSIONS p1,`
`Pessoa.VERSIONS p2`
`WHERE`
`p1.isPredecessorOf(p2);`
	`SELECT ...`
	`FROM pessoa t1, pessoa t2,`
	`PredSucc t3`
	`WHERE ...`

Do mesmo modo, quando a consulta apresentar alguma função de navegação na hierarquia de herança (`isAscendantOf` ou `isDescendantOf`), a tabela `AscDesc` é acrescentada na cláusula `FROM` da consulta mapeada.

Seguindo todos esses passos, a cláusula `FROM` da consulta SQL está montada. Nos próximos exemplos, quando são apresentadas as construções das cláusulas `SELECT` e `WHERE`, essa cláusula já aparece construída.

Gerando a Cláusula `SELECT` do SQL

Nesse ponto do mapeamento, todas as tabelas necessárias para a consulta já estão inseridas na cláusula `FROM`. Para gerar a cláusula `SELECT` da SQL é necessário percorrer as cláusulas `SELECT` e `FROM` da TVQL. A cláusula `FROM` é verificada para

saber em quais tabelas estão os atributos a serem retornados, e a cláusula `SELECT` para verificar as características do que deve ser obtido na consulta (temporal ou não).

A cláusula `SELECT` é mapeada verificando em quais tabelas estão as propriedades selecionadas. Considerando as consultas onde a seleção é feita sobre os dados atuais, as propriedades são mapeadas para as tabelas principais. No entanto, se o retorno da consulta é histórico, as tabelas auxiliares participam da seleção.

No mapeamento da cláusula `SELECT` as variações são menores, mas não menos importantes, sendo necessárias algumas considerações descritas a seguir.

Caso 7. (`SELECT` e a palavra reservada `EVER`). As consultas que possuem a palavra reservada `EVER` na cláusula `SELECT` e atributos temporais, quando mapeadas, retornam o atributo `value` da tabela auxiliar. Por exemplo, a consulta a seguir recupera o histórico dos nomes de todas as versões de `Pessoa`:

TVQL	SQL (`SELECT`)

`SELECT EVER nome`
`FROM Pessoa.VERSIONS;`
	`SELECT t1.value`
	`FROM pessoaNome t1`
	`WHERE ...`

Quando a palavra `EVER` não aparece na consulta ou aparece somente na cláusula `WHERE`, o retorno da consulta é o valor atual do atributo. Portanto, a cláusula `SELECT` da SQL recebe o próprio atributo que se encontra na tabela principal. Tal que, para retornar os nomes atuais de todas as versões de pessoas, a sintaxe TVQL pode ser escrita conforme a apresentado a seguir:

TVQL	SQL (`SELECT`)

`SELECT nome`
`FROM Pessoa.VERSIONS;`
	`SELECT t1.nome`
	`FROM pessoa t1`
	`WHERE ...`

Para atributos não temporais, o mapeamento é trivial, pois esses só existem na tabela principal, onde foram criados no momento da geração das tabelas no banco de dados (mapeamento das classes do modelo).

Caso 8. (`SELECT` com propriedades temporais). Propriedades temporais podem estar presentes nas cláusulas `SELECT` e `WHERE` da consulta TVQL. No primeiro caso, essas propriedades são substituídas pelos respectivos rótulos temporais, conforme demonstrado na tabela 5.3.

TABELA 5.3 – Mapeamento dos rótulos temporais

TVQL	SQL
`tInterval`	`char(itTime) || '..' || char(ftTime)`
`vInterval`	`char(ivTime) || '..' || char(fvTime)`
`tiInstant`	`ItTime`
`tfInstant`	`FtTime`
`viInstant`	`IvTime`
`vfInstant`	`FvTime`
`iLifetime`	`TabelaAlive.ivTime`
`fLifetime`	`TabelaAlive.fvTime`

Para exemplificar o mapeamento dessas propriedades, a consulta a seguir retorna os intervalos de validade dos nomes de todas as versões de pessoas:

```
SELECT DISTINCT nome.vInterval
FROM Pessoa.VERSIONS;
```

E o seu mapeamento para SQL é escrito como:

```
SELECT DISTINCT
       char(t2.ivTime) ||'..'|| char(t2.fvTime)
FROM pessoa t1, pessoaNome t2
WHERE ...
```

Das propriedades apresentadas na tabela 5.3, `iLifetime` e `fLifetime` são propriedades definidas para a recuperação dos tempos de vida inicial e final dos objetos e versões. Essa característica é representada pelos tempos de validade inicial e final do atributo temporal `alive` quando esse possui o valor `true`. Essas propriedades existem apenas para deixar transparente para o usuário a implementação dos tempos de vida, bem como a recuperação dos rótulos temporais dos atributos.

Caso 9. (`SELECT` com propriedades para recuperação de versões). Na TVQL foi definido um conjunto de propriedades para que o controle de objeto versionado fosse consultado de maneira transparente. Essas propriedades podem ser utilizadas na cláusula `SELECT` da consulta, bem como no `WHERE`.

No caso da cláusula `SELECT` essas propriedades podem ser retornadas, desde que na SQL sejam feitas as respectivas equivalências entre a propriedade recuperada com o atributo que representa essa propriedade, de acordo com a tabela 5.4.

TABELA 5.4 – Mapeamento das propriedades de versionamento

TVQL	SQL
configurationCount	VOC.configCount
currentVersion	VOC.currentVersion
firstVersion	VOC.firstVersion
lastVersion	VOC.lastVersion
nextVersionNumber	VOC.nextVNumber
userCurrentFlag	VOC.userCurrentF
versionCount	VOC.versionCount

Por exemplo, a próxima consulta retorna as últimas versões de `pessoa`:

TVQL	SQL (SELECT)
SELECT p.lastVersion FROM Pessoa.VERSIONS p;	SELECT **t2.lastVersion** FROM Pessoa t1, VOC t2 WHERE ...

Das propriedades mencionadas na tabela 5.4, somente `nextVersionNumber` não é temporal. Nesse contexto, quando a palavra `EVER` estiver na cláusula `SELECT` sabe-se que a consulta é sobre o histórico dos atributos temporais. Portanto, a coluna `value` da tabela auxiliar do atributo temporal deve ser colocada na cláusula `SELECT` da SQL, como mostrado na consulta abaixo:

TVQL	SQL (SELECT)
SELECT EVER lastVersion FROM Pessoa.VERSIONS;	SELECT **t3.value** FROM Pessoa t1, VOC t2, VOClastVersion t3 WHERE ...

Neste estágio do mapeamento as cláusulas SELECT e FROM da SQL estão completamente especificadas, portanto, na próxima seção as consultas são mostradas com seus respectivos mapeamentos completos.

5.3.2 Condições especiais

Depois de mapeadas as cláusulas FROM e SELECT, o próximo passo é fazer o mapeamento da cláusula WHERE da TVQL. Para isso, é feita a análise e a tradução de todas as propriedades e funções definidas na TVQL mencionadas na cláusula WHERE da consulta. Além disso, é necessário analisar também a cláusulas FROM da TVQL para verificar se alguma junção é necessária.

Nesta seção especificamente, estão descritas as regras que compõem a construção da cláusula WHERE da SQL.

Caso 10. (condições especiais e junções). No caso das classes temporais versionadas, se a função VERSIONS não está especificada, a consulta deve considerar apenas os objetos versionados e versões atuais. Sendo assim, a tabela VOC é incluída na cláusula FROM, e uma condição especial é acrescentada na cláusula WHERE da consulta SQL:

```
(TabelaPrincipal.refVOC is null)
OR(TabelaPrincipal.refVOC =VOC.tvoid
AND TabelaPrincipal.tvoid = VOC.currentVersion)
```

Exemplificando essa restrição, a consulta que retorna os nomes atuais das versões atuais de pessoa (SELECT nome FROM pessoa), considerando que a classe Pessoa é temporal versionada e que o atributo nome é temporal. Nesse ponto do mapeamento, as cláusulas FROM e SELECT estão completamente mapeadas, e a cláusula WHERE possui sua primeira restrição:

```
SELECT t1.nome
FROM pessoa t1, VOC t2
WHERE (t1.refVOC is null)
   OR (t1.refVOC = t2.tvoid
       AND t1.tvoid = t2.currentVersion)
```

Nas ocasiões em que a consulta apresenta a função VERSIONS e possui alguma função de navegação na hierarquia de derivação ou alguma propriedade definida para controle de objetos versionado, a tabela VOC já foi inserida na cláusula FROM da SQL. Portanto, ao invés da condição anterior, a cláusula WHERE possui a junção da tabela principal com a tabela VOC, conforme descrito a seguir:

```
TabelaPrincipal.refVOC =VOC.tvoid
```

Uma vez especificadas na cláusula FROM todas as tabelas principais e auxiliares que fazem parte da SQL, é necessário acrescentar as respectivas junções na cláusula

WHERE. Sendo assim, para cada tabela auxiliar existente é necessário acrescentar na cláusula WHERE a seguinte linha de comando:

```
TabelaPrincipal.tvoid = TabelaAuxiliar.tvoid
```

Por *default*, nas consultas temporais, devem ser acrescentadas as seguintes linhas na cláusula WHERE:

```
TabelaPrincipal.Alive = 'T'
TabelaAuxiliar.ftTime = '9999-12-31 00:00:00'
```

A condição `alive = 'T'` deve ser acrescentada, pois normalmente as consultas são realizadas objetos versionados e versões ativas e se o usuário não especificar na consulta outra condição, fica subentendido que ele quer realizar as consultas sobre as versões ativas.

A condição `TabelaAuxiliar.ftTime = '9999-12-31 00:00:00'` refere-se ao tempo de transação final do atributo temporal e é necessária para recuperar informações referentes ao estado atual da base (considerando dados temporais cujo tempo de transação final está em aberto).

A menos que, na consulta TVQL estejam explícitos qualquer uma das condições:

```
Classe.isAlive = false
Classe.isDeactivated [At]
```

Ou ainda, quando alguma condição específica possuir alguma das seguintes propriedades temporais: tInterval e ftTime. Quando uma delas estiver na cláusula WHERE significa que já está sendo feita uma restrição ao limite final do tempo de transação do atributo temporal em questão.

Seguindo as regras mencionadas anteriormente, a consulta aos nomes atuais das versões atuais de pessoas está completa, conforme mostrado abaixo:

TVQL	**SQL (WHERE)**

`SELECT nome`
`FROM Pessoa;`
	`SELECT t1.nome`
	`FROM pessoa t1, VOC t2`
	`WHERE (t1.refVOC is null)`
	`  OR (t1.refVOC = t2.tvoid`
	`  AND t1.tvoid = t2.currentVersion)`
	`  AND t1.Alive = 'T';`

Essa consulta, apesar de ser bastante simples, precisa das restrições mencionadas acima por se tratar de uma consulta a uma classe temporal versionada (Pessoa). Caso o mesmo enunciado fosse alterado para o histórico do nome das versões de pessoa, as regras mencionadas até o momento contemplam todo o mapeamento. Desse modo, a consulta seria escrita e mapeada conforme apresentado a seguir:

TVQL	**SQL (WHERE)**

`SELECT EVER nome`
`FROM Pessoa.VERSIONS;`
	`SELECT t2.value`
	`FROM pessoa t1, pessoaNome t2`
	`WHERE t1.tvoid = t2.tvoid`
	`  AND t1.alive = 'T'`
	`  AND t2.tfTime = '9999-12-31 00:00:00';`

Quando a consulta for atual, mas alguma tabela auxiliar estiver presente na SQL, é necessário acrescentar outra condição para filtrar somente as informações válidas atualmente. A condição a ser acrescentada é:

```
TabelaAuxiliar.ivTime <= now
TabelaAuxiliar.fvTime >= now
```

A consulta sobre a classe `Pessoa` que retorna o nome e seu intervalo de validade, das versões atuais das pessoas que tem seu nome iniciado pela letra "A", é um exemplo que justifica o acréscimo dessa restrição na construção da cláusula `WHERE` da SQL.

TVQL	**SQL (WHERE)**

`SELECT nome,`
`  nome.vInterval`
`FROM Pessoa`
`WHERE nome like 'A%';`
	`SELECT t1.nome`
	`FROM pessoa t1, pessoaNome t2`
	`WHERE t1.tvoid = t2.tvoid`
	`  AND t1.alive = 'T'`
	`  AND t2.tfTime = '9999-12-31 00:00:00'`
	`  AND t2.ivTime <= now`
	`  AND t2.fvTime >= now;`

Essa condição compara a data atual com os intervalos de validade do atributo temporal. Além disso, só é acrescentada na sintaxe SQL se o aspecto temporal do atributo não estiver sendo considerado na consulta TVQL.

5.3.3 Mapeamento de restrições temporais

As restrições temporais mencionadas na consulta TVQL merecem mapeamentos específicos, conforme apresentado nessa seção.

Caso 11. (restrições sem tempo nem versões). Com relação às restrições normais, ou seja, sem tempo e versões, o mapeamento é comum, considerando somente as tabelas principais da aplicação (seja o atributo temporal ou não).

Por exemplo, para consultar todos os `nomes` e `cpfs` de todas as versões de `pessoa` que começam com o número 1 cujo nome comece também com a letra "A", o comando TVQL e o mapeamento para SQL é definido conforme mostrado a seguir:

TVQL	**SQL (WHERE)**

`SELECT cpf, nome`
`FROM Pessoa.VERSIONS`
`WHERE cpf like '1%' AND`
`  Nome like 'A%';`
	`SELECT t1.cpf, t1.nome`
	`FROM Pessoa t1`
	`WHERE t1.alive = 'T'`
	`  t1.cpf like '1%' AND`
	`  t1.nome like 'A%';`

Caso 12. (restrições envolvendo atributos temporais – com a palavra reservada `EVER` na consulta). Nas consultas com a palavra `EVER` em que as restrições apresentam atributos temporais, na cláusula `WHERE` a comparação é feita com o atributo `value` da tabela auxiliar. Por exemplo, a consulta sobre o histórico dos nomes de todas as versões das pessoas que algum dia tiveram seus nomes iniciados por "A", é escrita como:

TVQL	SQL (WHERE)
SELECT EVER nome FROM Pessoa.VERSIONS WHERE nome like 'A%';	SELECT t2.value FROM pessoa t1, pessoaNome t2 WHERE t1.tvoid = t2.tvoid AND t1.alive = 'T' AND t2.ftTime = '9999-12-31 00:00:00' AND **t2.value like 'A%';**

A consulta anterior considera o histórico das propriedades que foram selecionadas, bem como das restrições sobre os dados, pois a palavra `EVER` aparece na cláusula `SELECT`. Por outro lado, a palavra reservada `EVER` pode aparecer somente na cláusula `WHERE`, indicando que o retorno dos dados é atual e, a partir da inclusão da palavra `EVER`, a restrição passa a ser temporal. Refazendo a consulta anterior, o objetivo agora é retornar o `nome` e `salario` atual de todas as versões das pessoas que algum dia tiveram seu salário superior a 2000. O comando TVQL é escrito como:

TVQL	SQL (WHERE)
SELECT DISTINCT nome, salario FROM Pessoa.VERSIONS WHERE EVER salario > 2000;	SELECT DISTINCT t1.nome, t1.salario FROM pessoa t1, pessoaSalario t2 WHERE t1.tvoid = t2.tvoid AND t1.alive = 'T' AND t2.ftTime = '9999-12-31 00:00:00' **AND t2.value > 2000;**

O que muda dessa consulta para a anterior é que o retorno é atual (cláusula `SELECT`), considerando o atributo da tabela principal. Por se tratar de uma restrição temporal a comparação feita sobre o salário utiliza também o atributo `value` da tabela auxiliar `pessoaSalario`.

Outra variação do uso da palavra reservada `EVER` refere-se à função `PRESENT` na construção da consulta TVQL. Nesse caso, a consulta possui a característica temporal (palavra `EVER`) seguida da palavra `PRESENT` na cláusula `WHERE` para anular a condição temporal especificada anteriormente. Sendo assim, caso a comando `PRESENT` esteja especificado no comando TVQL, a cláusula `FROM` deve conter as tabelas principais e auxiliares. Dentro da função `PRESENT`, os atributos temporais devem ser mapeados para a coluna equivalente da tabela principal. Por outro lado, os atributos temporais que estiverem fora da função, devem ser mapeados para a coluna `value` da tabela auxiliar do atributo em questão.

Para exemplificar o uso da função `PRESENT` na cláusula `WHERE`, o enunciado da consulta anterior é alterado para que retorne o histórico dos nomes e os respectivos intervalos de validade das pessoas que atualmente recebem um salário superior a 2000, da seguinte maneira:

TVQL	SQL (WHERE)
SELECT EVER nome, nome.vInterval FROM Pessoa.VERSIONS WHERE PRESENT(salario > 2000);	SELECT t2.value, char(t2.ivTime) || '..' || char(t2.fvTime) FROM pessoa t1, pessoaNome t2 WHERE t1.tvoid = t2.tvoid AND t1.alive = 'T' AND t2.ftTime = '9999-12-31 00:00:00' AND **t1.salario > 2000;**

Em resumo, quando os atributos temporais estão contidos na consulta TVQL e a restrição é temporal, usa-se o atributo `value` da tabela auxiliar. Caso contrário, o atributo correspondente da tabela principal é usado.

Caso 13. (restrições envolvendo operadores de comparação e rótulos temporais). Os operadores de comparação herdados da SQL podem ser usados juntamente com as propriedades temporais e, quando isso acontece, essas propriedades são mapeadas para os rótulos das tabelas auxiliares que representam os atributos temporais conforme apresentado na tabela 5.5.

TABELA 5.5 – Mapeamento das propriedades temporais

TVQL	SQL
P.tiInstant	TabelaP.itTime
P.tfInstant	TabelaP.ftTime
P.viInstant	TabelaP.ivTime
P.vfInstant	TabelaP.fvTime

Exemplificando a utilização desses operadores e propriedades temporais na cláusula `WHERE`, a consulta abaixo retorna o histórico dos valores dos salários de todas as versões das pessoas que algum dia receberam salários superiores a 2000, conforme os dados armazenados na base até 31/12/2000:

```
SELECT EVER salario
FROM Pessoa.VERSIONS
WHERE salario.tfInstant <= '31/12/2000'
  AND salario > 2000;
```

Segundo as regras mencionadas até o momento, o mapeamento dessa consulta está completo, conforme mostrado abaixo:

```
SELECT t2.value
FROM Pessoa t1, PessoaSalario t2
WHERE t1.tvoid = t2.tvoid
  AND t1.alive = 'T'
  AND t2.value > 2000
  AND t2.ftTime <= '2000-12-31 00:00:00';
```

Além dos operadores normais da SQL, novos operadores foram definidos na TVQL. Dentre eles, estão os operadores para intervalos que no momento do mapeamento devem ser traduzidos segundos algumas regras. O intervalo de validade ou de transação pertence a uma propriedade temporal. Quando essa propriedade é mapeada, uma tabela auxiliar é criada para armazenar o histórico da propriedade juntamente com seus rótulos temporais. Esses rótulos são combinados e comparados entre si para realizar o mapeamento dos operadores de intervalo.

A figura 5.3 ilustra o funcionamento desses operadores, através da comparação realizada entre um intervalo representado pelos instantes `i1` e `f1` e outro intervalo `i2` e `f2` [MOR 2001b]. No operador `INTERSECT`, um intervalo intersecciona qualquer um dos pontos do outro intervalo; no operador `OVERLAP` o primeiro intervalo intersecciona os dois pontos do segundo e; no operador `EQUAL` os intervalos têm que ser idênticos.

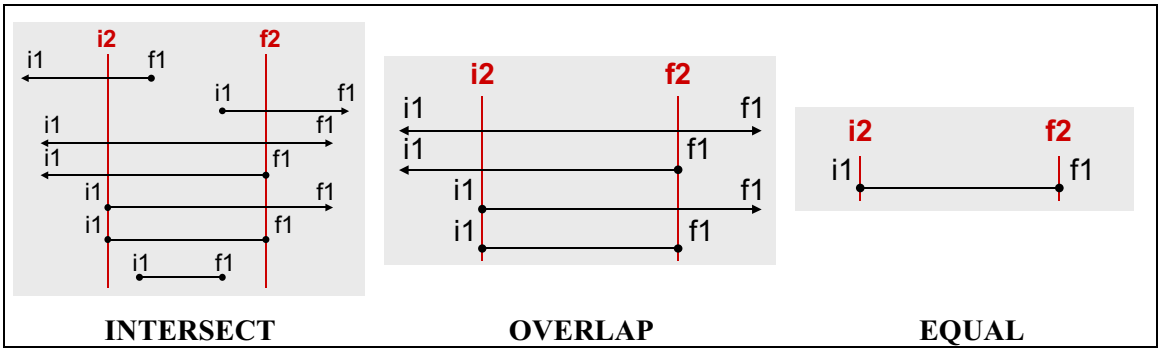

FIGURA 5.3 – Operadores de comparação para intervalos

A tabela 5.6, mostra como é mapeado cada um dos operadores para intervalos, considerando o intervalo de transação de uma propriedade comparado ao intervalo de transação de outra propriedade.

TABELA 5.6 – Mapeamento dos operadores de comparação para intervalos

TVQL	SQL

`p1.tInterval INTERSECT p2.tInterval`	`TabelaP1.itTime <= tabelaP2.ftTime`
	`AND`
	`TabelaP1.ftTime >= tabelaP2.itTime`

`p1.tInterval OVERLAP p2.tInterval`	`TabelaP1.itTime <= tabelaP2.itTime`
	`AND`
	`TabelaP1.ftTime >= tabelaP2.ftTime`

`p1.tInterval EQUAL p2.tInterval`	`TabelaP1.itTime = tabelaP2.itTime`
	`AND`
	`TabelaP1.ftTime = tabelaP2.ftTime`

Para mapear uma consulta que, ao invés de apresentar intervalo de transação em sua composição, apresente o intervalo de validade, basta substituir os rótulos `itTime` por `ivTime` e `ftTime` por `fvTime`.

É importante salientar que o `INTERSECT` é um dos mais importantes operadores de comparação e imprescindível quando uma consulta apresentar mais de um atributo temporal e o histórico de mais de um está envolvido na consulta. Nesses casos, a interseção dos intervalos de validade é necessária para recuperar apenas os valores válidos ao mesmo tempo. Para exemplificar essa definição, a figura 5.4 representa a evolução temporal do atributo `rua` e `salario` da classe `pessoa`, bem como as datas das atualização dos seus valores.

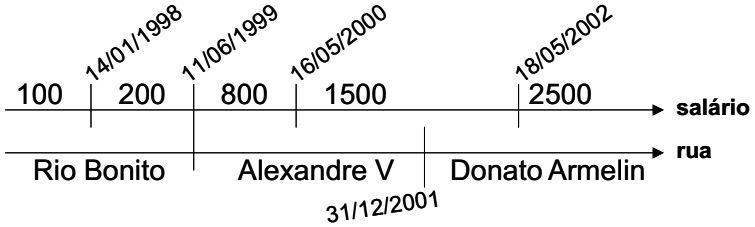

FIGURA 5.4 – Justificativa do uso do `INTERSECT`

Supondo que se queira recuperar o histórico da rua e do salário de todas as pessoas (supondo que `salario` e `rua` são temporais). A princípio, a consulta TVQL teria a seguinte sintaxe:

```
SELECT EVER rua, salario
FROM Pessoa.VERSIONS;
```

Quando aplicada a consulta anterior sobre os dados da figura 5.4, o retorno seria conforme exibido na tabela 5.7, ou seja, um produto cartesiano entre os rótulos temporais. No entanto, o retorno deveria ser somente o que está destacado na tabela 5.7.

TABELA 5.7 – Retorno da consulta sem o `INTERSECT`

rua	salario
Rio Bonito	100
Rio Bonito	200
Rio Bonito	800
Rio Bonito	1500
Rio Bonito	2500
Alexandre V	100
Alexandre V	200
Alexandre V	800
Alexandre V	1500
Alexandre V	2500
Donato Armelin	100
Donato Armelin	200
Donato Armelin	800
Donato Armelin	1500
Donato Armelin	2500

Como o objetivo é retornar somente as ruas que eram válidas ao mesmo tempo dos salários, é necessário adicionar o comando `INTERSECT` ao comando TVQL. Portanto, a consulta é reescrita para:

```
SELECT EVER rua, salario
FROM Pessoa.VERSIONS
WHERE nome.vInterval INTERSECT rua.vInterval;
```

Mapeando o `INTERSECT` juntamente com as regras definidas até o momento, o comando SQL é apresentado como segue:

```
SELECT t2.value, t3.value
FROM pessoa t1, pessoaRua t2, pessoaSalario t3
WHERE t1.tvoid = t2.tvoid
  AND t1.tvoid = t3.tvoid
  AND t1.alive = 'T'
  AND t2.ftTime = '9999-12-31 00:00:00'
  AND t3.ftTime = '9999-12-31 00:00:00'
  AND (t2.ivTime <= t3.fvTime
       AND t2.fvTime >= t3.ivTime);
```

Inicialmente, a idéia era deixar o operador INTERSECT implícito no mapeamento, mas se assim fosse, um universo muito grande de restrições seriam necessárias para realizar o mapeamento da TVQL corretamente. Algumas consultas teriam que implementar o INTERSECT, enquanto que outras não, dificultando o estabelecimento de uma regra. Por esta razão, caso exista mais de um atributo temporal em uma consulta que recupere o histórico, o operador INTERSECT deve estar explícito no comando TVQL, caso contrário o retorno dessa consulta resulta em um produto cartesiano.

Caso 14. (restrições envolvendo operadores de comparação para instantes e intervalos). Outra restrição de dados usada na TVQL é através do uso dos operadores de comparação para instantes e intervalos (BEFORE, INTO e AFTER).

Conforme mostrado na figura 5.5, o operador BEFORE verifica se um instante ou intervalo antecede completamente um intervalo; já o operador INTO verifica se o instante ou intervalo está contido em um intervalo e; por último o operador AFTER verifica se o instante ou intervalo está após um intervalo.

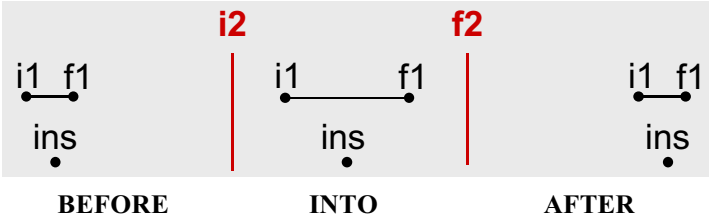

FIGURA 5.5 – Operadores de comparação para instantes e intervalos

No caso do uso desses operadores para comparar um instante com o intervalo, é necessário comparar os rótulos temporais do atributo em questão com o instante solicitado, fazendo a equivalência entre a propriedade temporal com seu respectivo rótulo, conforme a tabela 5.8.

TABELA 5.8 – Mapeamento dos operadores BEFORE, INTO e AFTER(instante)

TVQL	SQL
instante BEFORE p2.tInterval	instante < TabelaP2.itTime
instante INTO p2.tInterval	instante > TabelaP2.itTime AND instante < TabelaP2.ftTime
instante AFTER p2.tInterval	instante > TabelaP2.ftTime

Com relação aos operadores BEFORE, INTO e AFTER quando utilizados para comparar intervalos, são mapeados para a SQL conforme mostrado na tabela 5.9.

TABELA 5.9 – Mapeamento dos operadores BEFORE, INTO e AFTER(intervalo)

TVQL	SQL
p1.tInterval BEFORE p2.tInterval	TabelaP1.ftTime < TabelaP2.itTime
p1.tInterval INTO p2.tInterval	TabelaP1.itTime > TabelaP2.itTime AND TabelaP1.ftTime < TabelaP2.ftTime
p1.tInterval AFTER p2.tInterval	TabelaP1.itTime > TabelaP2.ftTime

O mapeamento da consulta que recupera o `tvoid` e o `salario` atual das versões de pessoa, que algum dia receberam salários superiores a 2000, antes de 11/06/2002.

```
SELECT tvoid, salario
FROM Pessoa.VERSIONS
WHERE EVER salario > 2000
  AND viInstant BEFORE '11/06/2002';
```

Traduzindo para a SQL a implementação é:

```
SELECT t1.tvoid, t1.salario
FROM pessoa t1, pessoaSalario t2
WHERE t1.tvoid = t2.tvoid
  AND t1.alive = 'T'
  AND t2.ftTime = '9999-12-31 00:00:00'
  AND t2.value > 2000
  AND t2.ivTime < '2002-06-11 00:00:00';
```

5.3.4 Mapeamento das restrições sobre versões

A partir da definição feita na cláusula `FROM` sobre quais objetos e versões devem ser consultados, o usuário pode utilizar na cláusula `WHERE` as funções e propriedades para a recuperação de informações sobre versões e objetos versionados. Nesse caso, seguem as considerações para mapear os aspectos de versões nas restrições da consulta (cláusula `WHERE`).

As funções da TVQL para manipulação de versões estão divididas em: (i) funções que recuperam o estado; (ii) funções para navegação na hierarquia de derivação e; (iii) funções para navegação na hierarquia de herança. Para cada um desses tipos algumas considerações são feitas.

Caso 15. (restrições envolvendo funções para recuperação de estado). Para cada estado que uma versão pode passar, existe uma função que retorna as versões do estado solicitado. Para mapear estas funções, no SQL a comparação é feita através do uso do atributo temporal `status` localizado na tabela principal. A comparação varia de acordo com a função utilizada no comando TVQL, conforme demonstrado na tabela 5.10.

TABELA 5.10 – Mapeamento das funções para recuperação do estado

Função	Mapeamento
IsWorking	Status = 'W'
IsStable	Status = 'S'
isConsolidated	Status = 'C'
IsDeactivated	Status = 'D'

Por exemplo, tem-se uma consulta que pretende retornar os nomes das versões de pessoa que atualmente possuam o estado estável. A consulta pode ser escrita como a seguir:

```
SELECT nome
FROM Pessoa.VERSIONS
WHERE pessoa.isStable;
```

Mapeando esta consulta, o comando SQL resulta em:

```
SELECT t1.nome
FROM Pessoa t1
WHERE t1.alive = 'T'
  AND t2.status = 'S';
```

A consulta anterior não considera os aspectos temporais, mas se a palavra `EVER` estivesse sendo considerada para a função, então no mapeamento a comparação não seria mais com o atributo `status` da tabela principal e sim com o atributo `value` da tabela auxiliar referente ao atributo temporal `status`.

Nesse caso, o retorno pretendido seria das versões que algum dia possuíram o `status` estável, então a consulta é reescrita para:

```
SELECT tvoid, nome
FROM Pessoa.VERSIONS
WHERE EVER pessoa.isStable;
```

E o mapeamento para a SQL é:

```
SELECT t1.tvoid, t1.nome
FROM Pessoa t1, PessoaStatus t2
WHERE t1.tvoid = t2. tvoid
  AND t1.alive = 'T'
  AND t2.ftTime = '9999-12-31 00:00:00'
  AND t2.value = 'S';
```

Dentre as funções que recuperam o estado, existem aquelas que recuperam o estado em um determinado instante. O mapeamento dessas funções está apresentado na tabela 5.11. Nesse caso, além de comparar o `status`, é necessário verificar também se o instante solicitado está contido nos instantes de validade inicial e final da tabela auxiliar que representa o `status`.

TABELA 5.11 – Funções com sufixo **`AT`** (recuperação do estado)

Função	**Mapeamento**
`isWorkingAt(instant)`	`TabelaAuxiliar.ivTime <= instant AND TabelaAuxiliar.fvTime >= instant AND TabelaAuxiliar.value = 'W'`
`isStableAt(instant)`	`TabelaAuxiliar.ivTime <= instant AND TabelaAuxiliar.fvTime >= instant AND TabelaAuxiliar.value = 'S'`
`isConsolidatedAt(instant)`	`TabelaAuxiliar.ivTime <= instant AND TabelaAuxiliar.fvTime >= instant AND TabelaAuxiliar.value = 'C'`
`isDeactivatedAt(instant)`	`TabelaAuxiliar.ivTime <= instant AND TabelaAuxiliar.fvTime >= instant AND TabelaAuxiliar.value = 'D'`

Sendo a consulta que retorne as versões que estavam consolidadas em "14/01/1999", sua sintaxe TVQL é escrita como:

```
SELECT nome, cpf
FROM pessoa.VERSIONS p
WHERE p.isStableAt('14/01/1999');
```

E o mapeamento para SQL é realizado conforme segue:

```
SELECT t1.nome, t1.cpf
FROM pessoa t1, pessoaStatus t2
WHERE t1.tvoid = t2.tvoid
  AND t1.alive = 'T'
  AND t2.ftTime = '9999-12-31 00:00:00'
  AND (t2.ivTime <= '1999-01-14 00:00:00'
  AND t2.fvTime >= '1999-01-14 00:00:00');
```

Caso 16. (restrições envolvendo funções para navegação na hierarquia de derivação). As funções para navegação na hierarquia de derivação estão representadas, em grande parte, pela tabela `VOC` gerada no momento do mapeamento das classes TVM. Adicionalmente, a tabela `PredSucc` está envolvida no mapeamento desse tipo de função, pois essa tabela está intimamente ligada a hierarquia de derivação quando a consulta envolve as funções que recuperam informações sobre sucessores ou predecessores. O mapeamento das funções principais desta categoria está apresentado na tabela 5.12.

TABELA 5.12 – Funções de navegação na hierarquia de derivação

Função	Consulta	Mapeamento
`Tabela.isFirst`	Atual	`VOC.firstVersion = Tabela.tvoid`
	Temporal	`Tabela.tvoid = VOCFirsVersion.value`
`Tabela.isLast`	Atual	`VOC.lastVersion = Tabela.tvoid`
	Temporal	`Tabela.tvoid = VOCLastVersion.value`
`Tabela.isSuccessorOf(TVO)`	Atual	`Tabela.tvoid = PredSucc.Successor` `AND TVO.tvoid = PredSucc.Predecessor`
	Temporal	`Tabela.tvoid = PredSucc.Successor` `AND TVO.tvoid = PredSucc.Predecessor`
`Tabela.isPredeccessorOf(TVO)`	Atual	`Tabela.tvoid = PredSucc.Predecessor` `AND TVO.tvoid = PredSucc.Successor`
	Temporal	`Tabela.tvoid = PredSucc.Successor` `AND TVO.tvoid = PredSucc.Predecessor`
`Tabela.isCurrent`	Atual	`VOC.currentVersion = Tabela.tvoid`
	Temporal	`Tabela.tvoid = VOCCurrentVersion.value`
`Tabela.isUserCurrent`	Atual	`VOC.userCurrentF = 'T'`
	Temporal	`Tabela.tvoid = VOCCurrentVersion.value` `AND VOCUserCurrentF.value = 'T'`
`Tabela.isConfiguration`	Atual	`Tabela.configuration = 'T'`
	Temporal	`TabelaAuxiliar.value = 'T'`

Para exemplificar o uso deste tipo de função, a consulta definida abaixo retorna o valor do `nome` das versões de pessoas que foram ou são primeiras versões:

```
SELECT nome
FROM Pessoa.VERSIONS
WHERE EVER pessoa.isFirst;
```

Em SQL a consulta é traduzida para:

```
SELECT t1.nome
FROM pessoa t1, VOC t2, VOCFirstVersion t3
WHERE t1.refVOC = t2.tvoid
  AND t2.tvoid = t3.tvoid
  AND t1.alive = 'T'
  AND t3.ftTime = '9999-12-31 00:00:00'
  AND t1.tvoid = t3.value;
```

A última condição expressa no mapeamento acima, faz uma junção indicando que a versão pode ter sido a primeira no passado ou ser atualmente a primeira versão (uma vez que os valores atuais também são mantidos na tabela auxiliar).

Caso a consulta fosse sem a palavra `EVER`, não seria necessário o uso da tabela `VOCFirstVersion`, pois a consulta seria somente à primeira versão atualmente. Nesse caso a comparação entre os `tvoids` seria substituída e a consulta, tanto em TVQL como em SQL seriam modificadas para:

TVQL	**SQL (WHERE)**

`SELECT nome`
`FROM Pessoa.VERSIONS p`
`WHERE p.isFirst;`
	`SELECT t1.nome`
	`FROM pessoa t1, VOC t2`
	`WHERE t1.refVOC = t2.tvoid`
	`  AND t1.alive = 'T'`
	`  AND t2.firstVersion = t1.tvoid;`

As funções que retornam predecessores ou sucessores merecem maior atenção, pois no seu mapeamento a tabela `PredSucc` está envolvida e, portanto, sua tradução para SQL apresenta algumas particularidades. Por exemplo, a consulta que retorna os nomes e o sucessor das versões de `Pessoa` é escrita e mapeada para SQL conforme mostrado a seguir:

TVQL	**SQL (WHERE)**

`SELECT p1.nome, p2.tvoid`
`FROM Pessoa.VERSIONS p1,`
`  Pessoa.VERSIONS p2`
`WHERE`
 	
`p1.isPredecessorOf(p2);`
	`SELECT t1.nome, t2.tvoid`
	`FROM pessoa t1, pessoa t2,`
	`  PredSucc t3`
	`WHERE t1.alive = 'T'`
	`  AND t2.alive = 'T'`
	`  AND t3.ftTime = '9999-12-31 00:00:00'`
	`  AND t1.tvoid = t3.predecessor`
	`  AND t2.tvoid = t3.successor;`

Ainda nas funções para navegação na hierarquia de derivação, quando envolvem propriedades temporais, essas funções possuem uma equivalente acrescida do sufixo `At`. Quando mapeadas, acessam a tabela auxiliar do atributo temporal e a comparação realizada segue a estrutura apresentada na tabela 5.13.

TABELA 5.13 – Funções com sufixo **AT** e de acesso a hierarquia de derivação

Função	Mapeamento

`Tabela.isFirstAt(instant)`	`VOCfirstVersion.ivTime <= instant`
	`AND VOCfirstVersion.fvTime >= instant`
	`AND Tabela.tvoid = VOCfirstVersion.value`

`Tabela.isLastAt(instant)`	`VOClastVersion.ivTime <= instant`
	`AND VOClastVersion.fvTime >= instant`
	`AND Tabela.tvoid = VOClastVersion.value`

`Tabela.isSuccessorOfAt (TVO, instant)`	`Tabela.tvoid = PredSucc.Successor`
	`AND TVO.tvoid = PredSucc.Predecessor`
	`AND PredSucc.ivTime <= instant`
	`AND PredSucc.fvTime >= instant`

`Tabela.isCurrentAt(instant)`	`VOCcurrentVersion.ivTime <= instant`
	`AND VOCcurrentVersion.fvTime >= instant`
	`AND Tabela.tvoid = VOCcurrentVersion.value`

`Tabela.isUserCurrentAt(instant)`	`VOCuserCurrentF.ivTime <= instant`
	`AND VOCuserCurrentF.fvTime >= instant`
	`AND VOCuserCurrentF.value = 'S'`

Exemplificando as funções de navegação na hierarquia de derivação e com o sufixo `At`, a consulta abaixo retorna o nome das versões de `Pessoa` que correspondiam às versões atuais em 16/05/2000:

TVQL	SQL (WHERE)

`SELECT nome`
`FROM Pessoa.VERSIONS p`
`WHERE`
`p.isCurrentAt('16/05/2000');`
	`SELECT t1.nome`
	`FROM pessoa t1, VOC t2,`
	`  VOCcurrentVersion t3`
	`WHERE t1.refVOC = t2.tvoid`
	`  AND t2.tvoid = t3.tvoid`
	`  AND t1.alive = 'T'`
	`  AND t3.ftTime = '9999-12-31 00:00:00'`
	`  AND t3.ivTime <= '2000-05-16 00:00:00'`
	`  AND t3.fvTime >= '2000-05-16 00:00:00';`

O sufixo `At` indica que a informação deve ser consultada conforme um determinado ponto no tempo de validade. Por essa razão, a tabela auxiliar deve estar presente na consulta SQL, para que seja possível comparar o instante desejado com os instantes de validade inicial e final da propriedade.

Caso 17. (restrições envolvendo funções para navegação na hierarquia de herança). As funções definidas para navegação na hierarquia de herança definidas na TVQL são: `isAscendantOf` e `isDescendantOf`.

Para implementar seus respectivos mapeamentos é necessário acessar a tabela `AscDesc` definida durante o mapeamento da hierarquia de classes do TVM. Nesse contexto, o mapeamento dessas funções equivale ao apresentado na tabela 5.14.

TABELA 5.14 – Funções para navegação na hierarquia de herança

Função	Mapeamento
`Tabela.isAscendantOf(TVO)`	`Tabela.tvoid = AscDesc.Ascendant` `AND TVO.tvoid = AscDesc.Descendant`

`Tabela.isAscendantOfAt(TVO,instant)`	`Tabela.tvoid = AscDesc.Ascendant`
	`AND TVO.tvoid = AscDesc.Descendant`
	`AND TVO.ivTime <= instant`
	`AND TVO.fvTime >= instant`

`Tabela.isDescendantOf(TVO)`	`Tabela.tvoid = AscDesc.Descendant` `AND TVO.tvoid = AscDesc.Ascendant`

`Tabela.isDescendantOfAt(TVO,instant)`	`Tabela.tvoid = AscDesc.Descendant`
	`AND TVO.tvoid = AscDesc.Ascendant`
	`AND TVO.ivTime <= instant`
	`AND TVO.fvTime >= instant`

Para demonstrar a implementação do mapeamento dessas funções, a consulta a seguir é escrita em TVQL e mapeada para SQL:

TVQL	SQL (WHERE)

`SELECT nome, p2.tvoid`
`FROM Pessoa.VERSIONS p1,`
`  Pessoa.VERSIONS p2`
`WHERE`
`p1.isAscendantOf(p2);`
	`SELECT t1.nome, t2.tvoid`
	`FROM pessoa t1, pessoa t2,`
	`  AscDesc t3`
	`WHERE t1.alive = 'T'`
	`  AND t2.alive = 'T'`
	`  AND t3.ftTime = '9999-12-31 00:00:00'`
	`  AND t1.tvoid = t3.Ascendant`
	`  AND t2.tvoid = t3.Descendant;`

Caso 18. (restrição envolvendo propriedades definidas para controle das versões e dos objetos versionados). Quanto às propriedades definidas para controle de objetos versionados, a tabela 5.15 apresenta a sua tradução para SQL, tanto para a restrição que considere as propriedades atuais, bem como para restrições temporal. A diferença é que, quando a condição é atual, a propriedade é mapeada para o respectivo atributo da tabela `VOC`. Caso contrário, é mapeada para o atributo `value` da tabela que representa o atributo temporal em questão.

TABELA 5.15 – Mapeamento das propriedades de versionamento

TVQL	SQL (atual)	SQL (temporal)
`configurationCount`	`VOC.configCount`	`VOCconfigCount.Value`
`currentVersion`	`VOC.currentVersion`	`VOCcurrentVersion.Value`
`firstVersion`	`VOC.firstVersion`	`VOCfirstVersion.Value`
`lastVersion`	`VOC.lastVersion`	`VOClastVersion.Value`
`nextVersionNumber`	`VOC.nextVNumber`	`VOC.nextVNumber`
`userCurrentFlag`	`VOC.userCurrentF`	`VOCuserCurrentF.Value`
`versionCount`	`VOC.versionCount`	`VOCversionCount.Value`

Para representar a tradução dessas propriedades TVQL para SQL, a consulta mostrada a seguir retorna a versão corrente de `Pessoa`, desde que possuam mais que 5 versões:

TVQL	**SQL (WHERE)**

`SELECT currentVersion`
`FROM Pessoa.VERSIONS`
`WHERE versionCount > 5;`
	`SELECT t2.currentVersion`
	`FROM Pessoa t1, VOC t2`
	`WHERE t1.refVoc = t2.tvoid`
	`  AND t1.alive = 'T'`
	`  AND` **`t2.versionCount > 5;`**

Do mesmo modo, caso a consulta envolvesse o aspecto temporal da propriedade, a comparação seria feita com a propriedade `value` da tabela auxiliar e esta participaria da cláusula `FROM` da SQL. Como por exemplo, a anterior reescrita com o acréscimo da palavra `EVER` na cláusula `WHERE`, é definida e mapeada a seguir:

TVQL	**SQL (WHERE)**

`SELECT currentVersion`
`FROM Pessoa.VERSIONS`
`WHERE`
`  EVER versionCount > 5;`
	`SELECT t2.currentVersion`
	`FROM Pessoa t1, VOC t2,`
	`  VOCversionCount t3`
	`WHERE t1.refVoc = t2.tvoid`
	`  AND t2.tvoid = t3.tvoid`
	`  AND t1.alive = 'T'`
	`  AND t3.ftTime = '9999-12-31 00:00:00'`
	`  AND t3.value > 5;`

5.4 Considerações Finais

Esse capítulo apresentou o mapeamento da linguagem TVQL para a SQL. Neste processo foram detectadas duas formas para realizar o mapeamento:

- traduzindo cada cláusula TVQL para a SQL, gerando parcialmente cada uma das cláusulas SQL; e
- realizando a construção de cada uma das cláusula SQL através da leitura das cláusulas TVQL. Essa alternativa foi escolhida, pois uma vez construída a cláusula SQL, esta não é mais modificada.

O detalhamento da alternativa escolhida para implementar o mapeamento foi dividido em quatro etapas e para cada etapa, foram avaliadas as diversas possibilidades para a construção das cláusulas da SQL. Em resumo, a tabela 5.16 apresenta o mapeamento da TVQL para SQL, detalhando as etapas para o mapeamento e os passos necessários para realizá-lo.

TABELA 5.16 – Resumo do mapeamento da TVQL para SQL

Classes, Atributos e Associações	
Possibilidades	**Detalhamento**
Caso1. `FROM` com e sem `VERSIONS`	Determina quando incluir ou não a tabela `VOC` na cláusula `FROM` da SQL.
Caso2. `FROM` sem `EVER`	Determina a construção da cláusula `FROM` da SQL, das consulta que não apresentam aspectos temporais (uso somente das tabelas principais).
Caso3. `FROM` com `EVER` e atributos temporais	Determina a presença das tabelas auxiliares na consulta SQL nas consultas com aspectos temporais.
Caso4. `FROM` com `EVER` e atributos temporais e não temporais	Construção da cláusula `FROM` da SQL, em consultas temporais com atributos temporais e não temporais.
Caso5. `FROM` com `EVER`, mas com propriedades temporais, operadores de comparação ou funções com sufixo `AT`	Determina quais tabelas auxiliares participam da consulta, mesmo que a palavra `EVER` não conste na TVQL.
Caso6. Funções de navegação na hierarquia de herança e funções referentes a sucessores e predecessores	Determina a inclusão das tabelas `PredSucc` e `AscDesc` na cláusula `FROM`.
Caso7. Consultas TVQL com e sem `EVER`	Determina se o retorno da consulta será buscado da tabela principal ou da auxiliar.
Caso8. `SELECT` com propriedades temporais	Mapeia as propriedades temporais da TVQL para os respectivos rótulos temporais
Caso9. `SELECT` com propriedades para recuperação de versões	Mapeia as propriedades de versionamento através do retorno do correspondente na tabela `VOC` ou na tabela auxiliar, caso a propriedade for temporal.
Condições Especiais	
Caso10. Condições especiais e junções	Inicia a construção da cláusula `WHERE` com restrições especiais que não foram descritas na TVQL.
Restrições Temporais	
Caso11. Restrições sem tempo nem versões	Monta a cláusula `WHERE` da SQL buscando os atributos nas tabelas principais.
Caso12. Restrições temporais com a palavra `EVER`	Monta a cláusula `WHERE` da SQL considerando o atributo `value` das tabelas auxiliares.
Caso13. Operadores de comparação e rótulos temporais	Mapeia os operadores de comparação somente para intervalos, juntamente com as propriedades temporais da cláusula `WHERE` da TVQL.
Caso14. Operadores de comparação para instantes e intervalos	Mapeia os operadores de comparação para instantes e intervalos.

Restrições sobre Versões	
Caso15. Restrições com funções para recuperação do estado	Traduz as funções para recuperação do estado das versões.
Caso16. Restrições com funções para navegação na hierarquia de derivação	Traduz as funções para navegação na hierarquia de derivação.
Caso17. Restrições com funções para navegação na hierarquia de herança	Traduz as funções para navegação na hierarquia de herança.
Caso18. Restrições com propriedades definidas para controle das versões e dos objetos versionados	Determina o mapeamento das propriedades de versionamento.

6 Implementação

Implementar um modelo de dados que estende o SQL com suporte a tempo é uma tarefa de alto custo e somente os maiores fabricantes de banco de dados podem financiar [JEN 99]. Nesse contexto, uma alternativa é prover suporte para aplicações de tempo e versões através de uma camada intermediária entre o SGBD existente e a aplicação (figura 6.1) [MOR 2001a].

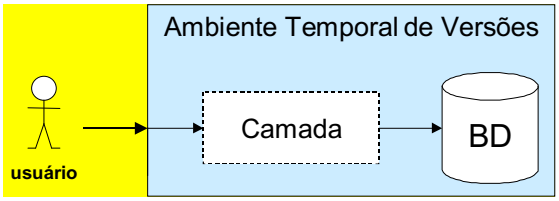

FIGURA 6.1 – Ambiente Temporal de Versões

Este capítulo resume o Ambiente Temporal de Versões que implementa o TVM sobre um banco de dados convencional. Especificamente, a interface para consultas TVQL proposta nessa dissertação é apresentada em detalhes. Por fim, um pequeno estudo de caso é mostrado, ilustrando exemplos de consultas que são executadas na interface.

6.1 Ambiente Temporal de Versões

Utilizando uma camada intermediária, é possível maximizar o reuso de tecnologias existentes e apresentar um SGBD com todas as características necessárias. O Ambiente Temporal de Versões proporciona ao usuário as funcionalidades básicas de um SGBD, divididas nos seguintes módulos:

- especificação de classes –especifica classes através da ferramenta de apoio ou através de um arquivo com as definições. O arquivo gerado em SQL pode ser executado na base;
- gerenciamento de classes – permite alterações nas definições das classes (atributos, relacionamentos, cardinalidades, ...);
- gerenciamento de dados – permite a instanciação de objetos e seu gerencimento;
- consulta a dados – permite a realização de consultas a base, obtendo os resultados em texto, tabela ou gráficos.

Todas essas funções são mapeada pela camada intermediária para o seu respectivo conjunto de comandos ou consultas SQL, conforme ilustrado na figura 6.2.

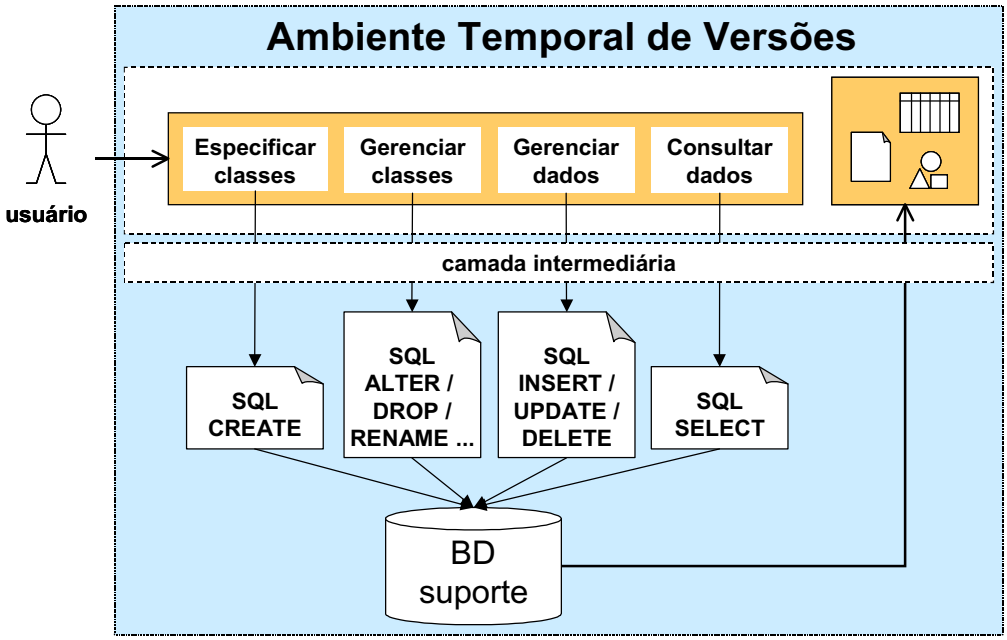

FIGURA 6.2 – Interações do usuário com o Ambiente [Moro 2001a]

Este trabalho propõe o mapeamento das classes do TVM para uma banco de dados relacional e o mapeamento da consulta TVQL para SQL. A seguir é detalhada a interface para consulta em TVQL.

6.2 A Interface de Consulta

Todos os exemplos apresentados neste capítulo foram executados a partir da interface de consulta implementada para a TVQL. Foi construído um protótipo para que fosse possível testar as consultas que foram mapeadas para a SQL. Inicialmente, este protótipo apresenta uma tela contendo uma lista de *aliases* existentes no computador, onde o usuário deve selecionar o *alias* do banco de dados que deseja consultar, conforme ilustrado na figura 6.3.

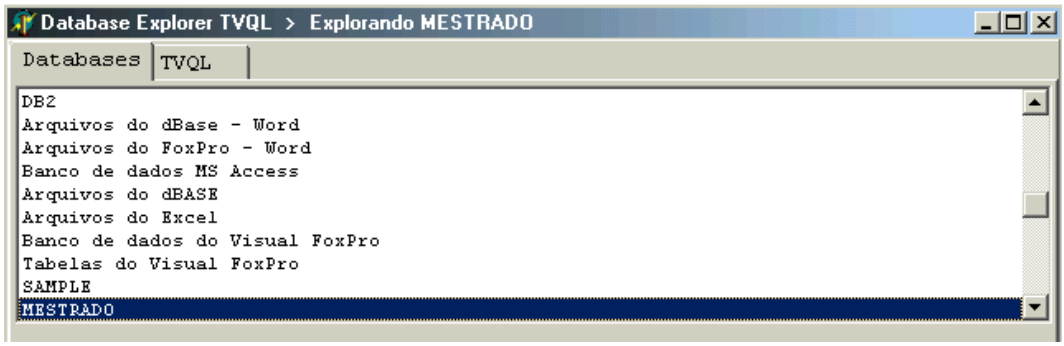

FIGURA 6.3 – Interface inicial do Protótipo da TVQL

Neste trabalho é implementado o mapeamento para o banco de dados DB2, no entanto todos os *aliases* estão disponíveis. A idéia é que nessa interface a consulta seja realizada em TVQL e o mapeamento se encarregue de mapear para a sintaxe do banco de dados selecionado.

Depois de escolher o *alias*, o usuário deve acionar a paleta “TVQL”, e uma nova tela ficará disponível para realizar as consultas TVQL. Como ilustrado na figura 6.4, na

parte superior da tela é digitado o comando TVQL e para executá-lo basta acionar o botão representado por um raio ou utilizar a tecla de atalho (CTRL+E). A consulta TVQL digitada pode ser exibida com seu respectivo mapeamento caso o usuário acionar o botão identificado pelo texto "SQL" ou pressionar CTRL+Q. Na parte inferior da tela é exibido o resultado da consulta expressa pelo comando TVQL.

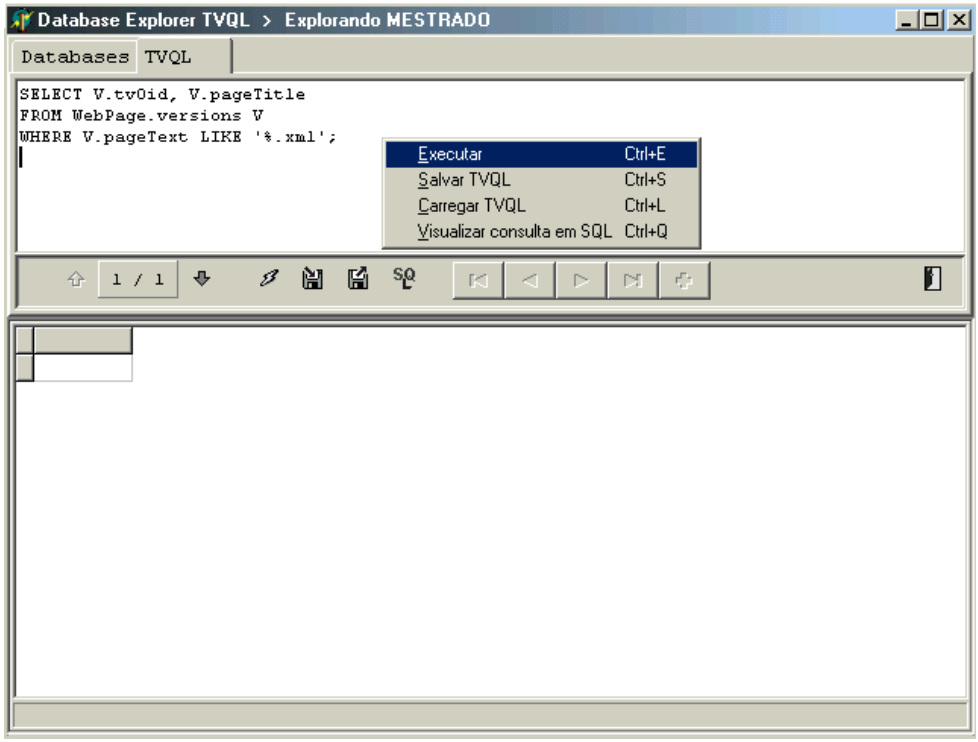

FIGURA 6.4 – Interface de consulta do protótipo

Na implementação desse protótipo pressupõe-se que o comando TVQL está escrito corretamente, ou seja, não está sendo realizada a análise léxica, sintática nem semântica da TVQL.

6.3 Estudo de Caso

Essa seção apresenta a modelagem de um estudo de caso sobre o qual são realizadas consultas com a TVQL. Para ilustrar os resultados das consultas precisa-se mostrar os dados da base. Adicionalmente, as consultas são geradas considerando o mapeamento proposto nessa dissertação.

6.3.1 A Aplicação

O estudo de caso utilizado neste trabalho é um resumo do apresentado por Moro, consistindo da modelagem de uma empresa de criação de *websites* [MOR 2001a]. Nessa empresa, além dos *sites* de seus clientes, são mantidas as páginas profissionais de seus empregados. Um *website* é composto de uma ou mais páginas, sendo uma delas a página inicial. Por exemplo, o cliente pode querer manter em páginas diferentes seus dados pessoais, uma relação de seus *sites* preferidos, e assim por diante, mas todas referenciadas a partir da mesma página principal.

Um *website* possui um padrão de página associado composto pela cor e imagem de fundo da página, 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.

A figura 6.5 apresenta apenas uma parte simplificada do diagrama de classes do modelo. As especificações do cliente e dos empregados que possuem os `websites` não são apresentadas. Além disso, as operações estão omitidas em todas as classes.

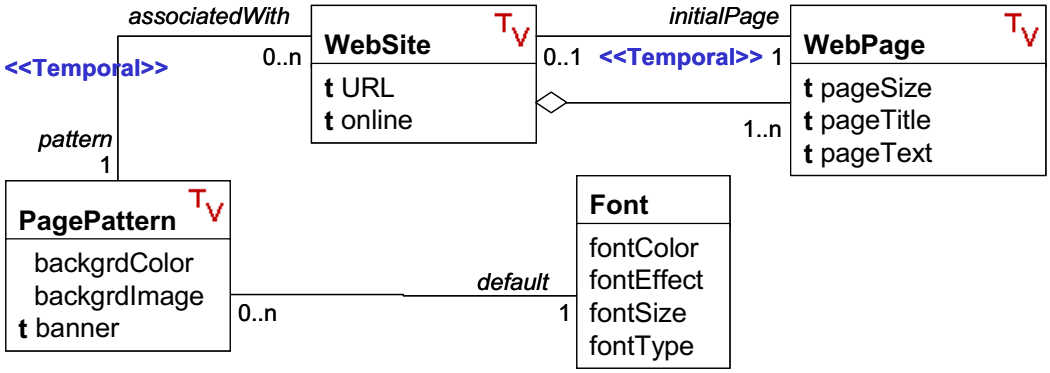

FIGURA 6.5 – Diagrama de classes do estudo de caso

As características de versão e tempo são usadas em:

- classes `WebSite` e `WebPage` – estas armazenam todas as alternativas do *site* e de suas páginas, além do histórico de seus atributos;
- classe `PagePattern` – armazena o histórico do atributo `banner`;
- relacionamento `pattern` (inverso `associatedWith`) – é considerado um relacionamento chave do modelo. Através dele a empresa troca o padrão dos *websites* de seus empregados para estarem de acordo com a estação do ano ou data comemorativa.
- relacionamento `initialPage` – armazena o histórico das páginas iniciais.

6.3.2 Dados

A ferramenta que implementa o Ambiente já proporciona a criação de tabelas no banco de dados relacional DB2 a partir da especificação do usuário. Considerando então 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 3.

Como primeira parte dos dados armazenados, são apresentadas as evoluções das instâncias de dois objetos versionados da classe `WebPage`.

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

Nesse contexto, a tabela 6.1 apresenta a evolução das versões da instância da funcionária de código 52. A página foi instanciada no dia dois de janeiro, contendo o texto base (“E52base.htm”). No dia dez de janeiro, foi derivada a versão com imagens (“E52gr.htm”) que foi atualizada dois dias depois por ultrapassar o limite de tamanho de 100Kb, estabelecido pela empresa (“E52gr2.htm”). No dia quinze de março, foi derivada a versão XML da página (“E52x.xml”).

Duas observações quanto às tabelas dos atributos temporais:

- para evitar excesso de informações, todas as datas 31/12/9999 estão omitidas, sendo consideradas o valor do rótulo temporal quando o mesmo está em branco;
- as linhas marcadas em cinza são aquelas inseridas na tabela para manter o histórico dos tempos de transação e validade.

TABELA 6.1 – Evolução do primeiro objeto de `WebPage`, tabela `WebPage`

tvoid	alive	configuration	status	PageTitle	pageText	PageSize	refVOC
101,22,1	true	false	S	Fulana de Tal	E52base.htm	7	101,22,null
101,22,2	true	false	W	Fulana de Tal	E52gr.htm	89	101,22,null
101,22,3	true	false	W	Fulana de Tal	E52x.xml	5	101,22,null

A tabela 6.2 apresenta a evolução das páginas da funcionária de código 78. Essa funcionária possui sua página base (“E78base.htm”) e sua página com imagens (“E78gr.htm”) como a funcionária anterior. Porém, no dia dezoito de março, foi armazenado o nome de casada dessa funcionária. Então, ocorreram a atualização do título da página com imagens (“Ciclana Xavier”) e a derivação de uma nova página base (“E78base2.htm”), pois a versão está no estado estável e não pode ser alterada.

A segunda versão desses objetos foi escolhida pelo usuário como a versão corrente. Quando o objeto versionado for consultado, os valores das segundas versões são retornados.

TABELA 6.2 – Evolução do segundo objeto de `WebPage`, tabela `WebPage`

tvoid	alive	configuration	status	pageTitle	pageText	pageSize	refVOC
125,22,1	true	False	S	Ciclana Saraiva	E78base.htm	6	125,22,null
125,22,2	true	False	W	Ciclana Xavier	E78gr.htm	82	125,22,null
125,22,3	true	False	W	Ciclana Xavier	E78base2.htm	6	125,22,null

Dados dos Metadados

Nos metadados, as informações de duas classes são fundamentais para o entendimento dos dados: `Class` e `Entity`, que armazenam respectivamente as informações das classes e entidades. Além dessas, a classe `Name` armazena os nomes (ou apelidos) das instâncias de alguns objetos, conforme definido pelo usuário.

As tabelas 6.3, 6.4 e 6.5 apresentam os dados da classe `WebPage` na tabela `Class`, das entidades das funcionárias de código 52 e 78 na tabela `Entity`, e os nomes das instâncias dessas funcionárias na classe `WebPage` na tabela `Name`.

TABELA 6.3 – Metadados, tabela `Class`

id	name	tempVers	specializ	visibility	root
22	Webpage	true	none	public	true

TABELA 6.4 – Metadados, tabela `Entity`

id	name
101	E52_Fulana
125	E78_Ciclana

TABELA 6.5 – Metadados, tabela `Name`

tvoid	name
101,22,1	E52_base
101,22,2	E52_grafico
101,22,3	E52_xml
125,22,1	E78_base
125,22,2	E78_grafico
125,22,3	E78_base2

Dados da Aplicação

As classes de aplicação temporais versionadas herdam os atributos de `Object`, `TemporalObject` e `TemporalVersion`. Essa subseção apresenta os dados, para cada funcionária, dos históricos dos atributos temporais herdados:

- `alive` – instanciado quando o objeto é criado (tabela 6.6);
- `status` – instanciado quando o objeto é criado e alterado quando novas versões são derivadas (tabela 6.7);
- `ascendant` e `descendant` – instanciado quando o objeto é criado, (tabela 6.8);
- `predecessor` e `successor` – instanciado quando o objeto é criado e alterado quando novas versões são derivadas (tabela 6.9).

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

TABELA 6.6 – Atributo temporal `alive`, tabela `WebPageAlive`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,1	true	02/01/2002		02/01/2002	
101,22,2	true	10/01/2002		10/01/2002	
125,22,1	true	12/02/2002		12/02/2002	
125,22,2	true	15/02/2002		15/02/2002	
101,22,3	true	15/03/2002		15/03/2002	
125,22,3	true	18/03/2002		18/03/2002	

TABELA 6.7 – Atributo temporal `status`, tabela `WebPageStatus`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,1	W	02/01/2002	09/01/2002	02/01/2002	
101,22,2	W	10/01/2002		10/01/2002	
101,22,1	W	10/01/2002		02/01/2002	09/01/2002
101,22,1	S	10/01/2002		10/01/2002	
125,22,1	W	12/02/2002	14/02/2002	12/02/2002	
125,22,2	W	15/02/2002		15/02/2002	
125,22,1	W	15/02/2002		12/02/2002	14/02/2002
125,22,1	S	15/02/2002		15/02/2002	
101,22,3	W	15/03/2002		15/03/2002	
125,22,3	W	18/03/2002		18/03/2002	

TABELA 6.8 – Ascendentes e descendentes, tabela `AscDesc`

seqAD	ascendant	descendant	itTime	ftTime	ivtime	FvTime
1	null	101,22,1	02/01/2002		02/01/2002	
2	101,22,1	null	02/01/2002		02/01/2002	
3	null	101,22,2	10/01/2002		10/01/2002	
4	101,22,2	null	10/01/2002		10/01/2002	
5	null	125,22,1	12/02/2002		12/02/2002	
6	125,22,1	null	12/02/2002		12/02/2002	
7	null	125,22,2	15/02/2002		15/02/2002	
8	125,22,2	null	15/02/2002		15/02/2002	
9	null	101,22,3	15/03/2002		15/03/2002	
10	101,22,3	null	15/03/2002		15/03/2002	
11	null	125,22,3	18/03/2002		18/03/2002	
12	125,22,3	null	18/03/2002		18/03/2002	

TABELA 6.9 – Predecessores e sucessores, tabela `PredSuc`

seqPS	predecessor	successor	itTime	ftTime	ivtime	fvTime
1	Null	101,22,1	02/01/2002		02/01/2002	
2	101,22,1	null	02/01/2002	09/01/2002	02/01/2002	
3	101,22,1	null	10/01/2001		02/01/2002	09/01/2002
4	101,22,1	101,22,2	10/01/2002		10/01/2002	
5	101,22,2	null	10/01/2002		10/01/2002	
6	null	125,22,1	12/02/2002		12/02/2002	
7	125,22,1	null	12/02/2002		12/02/2002	
8	125,22,1	null	15/02/2002		12/02/2002	14/02/2002
9	125,22,1	125,22,2	15/02/2002		15/02/2002	
10	125,22,2	null	15/02/2002		15/02/2002	
11	101,22,1	101,22,3	15/03/2002		15/03/2002	
12	101,22,3	null	15/03/2002		15/03/2002	
13	125,22,1	125,22,3	18/03/2002		18/03/2002	
14	125,22,3	null	18/03/2002		18/03/2002	

Agora, passando para os valores dos históricos dos três atributos de `WebPage`:

- `pageTitle` – instanciado quando o objeto é criado e alterado quando a funcionária 78 casa (altera as versões "125,22,2" e o objeto versionado "125,22,0", pois a segunda versão é a corrente) (tabela 6.10);

- `pageText` – instanciado quando o objeto é criado e alterado quando é necessário diminuir o tamanho da versão "101,22,2", atualizando o valor da versão corrente também ("101,22,0") (tabela 6.11);
- `pageSize` – instanciado quando o objeto é criado e alterado quando é necessário diminuir o tamanho da versão "101,22,2", atualizando o valor da versão corrente também ("101,22,0") (tabela 6.12).

TABELA 6.10 – Atributo temporal `pageTitle`, tabela `WebPagePageTitle`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,1	Fulana de Tal	02/01/2002		02/01/2002	
101,22,2	Fulana de Tal	10/01/2002		10/01/2002	
125,22,1	Ciclana Saraiva	12/02/2002		12/02/2002	
125,22,2	Ciclana Saraiva	15/02/2002	17/03/2002	15/02/2002	
101,22,3	Fulana de Tal	15/03/2002		15/03/2002	
125,22,2	Ciclana Saraiva	18/03/2002		02/01/2002	17/03/2002
125,22,2	Ciclana Xavier	18/03/2002		18/03/2002	
125,22,3	Ciclana Xavier	18/03/2002		18/03/2002	

TABELA 6.11 – Atributo temporal `pageText`, tabela `WebPagePageText`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,1	E52base.htm	02/01/2002		02/01/2002	
101,22,2	E52gr.htm	10/01/2002	11/01/2002	10/01/2002	
101,22,2	E52gr.htm	12/01/2002		10/01/2002	11/01/2002
101,22,2	E52gr2.htm	12/01/2002		12/01/2002	
125,22,1	E70base.htm	12/02/2002		12/02/2002	
125,22,2	E70gr.htm	15/02/2002		15/02/2002	
101,22,3	E52x.xml	15/03/2002		15/03/2002	
125,22,3	E70base2.htm	18/03/2002		18/03/2002	

TABELA 6.12 – Atributo temporal `pageSize`, tabela `WebPagePageSize`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,1	7	02/01/2002		02/01/2002	
101,22,2	115	10/01/2002	11/01/2002	10/01/2002	
101,22,2	115	12/01/2002		10/01/2002	11/01/2002
101,22,2	94	12/01/2002		12/01/2002	
125,22,1	6	12/02/2002		12/02/2002	
125,22,2	85	15/02/2002		15/02/2002	
101,22,3	5	15/03/2002		15/03/2002	
125,22,3	6	18/03/2002		18/03/2002	

A tabela 6.13 apresenta as instâncias dos controles dos objetos das duas funcionárias 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 6.13 – Controle dos objetos versionados, tabela VOC

tvoid	config Count	Current Version	first Version	last Version	nextV Number	user CurrentF	version Count
101,22,null	0	101,22,2	101,22,1	101,22,3	4	false	3
125,22,null	0	125,22,2	125,22,1	125,22,3	4	false	3

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

- atributos `configurationCount` e `firstVersion` – são armazenados na instanciação e seus valores não são alterados no exemplo (não é gerada nenhuma), conforme apresentado em tabela 6.14 e tabela 6.15;
- `userCurrentF` – é armazenado na instanciação e seus valores são alterados quando o usuário estabelece a segunda versão do objeto como a corrente, conforme apresentado na tabela 6.16;
- atributos `currentVersion`, `lastVersion` e `versionCount` – são armazenados na instanciação e alterados nas derivações das terceiras versões, conforme apresentado em tabela 6.17, tabela 6.18 e tabela 6.19.

TABELA 6.14 – Controle dos objetos versionados, tabela VOCconfigCount

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	0	02/01/2002		02/01/2002	
125,22,null	0	12/02/2002		12/02/2002	

TABELA 6.15 – Controle dos objetos versionados, tabela VOCfirstVersion

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	101,22,1	02/01/2002		02/01/2002	
125,22,null	125,22,1	12/02/2002		12/02/2002	

TABELA 6.16 – Controle dos objetos versionados, tabela VOCuserCurrentF

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	false	02/01/2002	09/01/2002	02/01/2002	
125,22,null	false	12/02/2002	14/02/2002	12/02/2002	
101,22,null	false	10/01/2002		02/01/2002	09/01/2002
101,22,null	true	10/01/2002		10/01/2002	
125,22,null	false	15/02/2002		12/02/2002	14/02/2002
125,22,null	true	15/02/2002		15/02/2002	

TABELA 6.17 – Controle dos objetos versionados, tabela VOCcurrentVersion

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	101,22,2	02/01/2002		02/01/2002	
125,22,null	125,22,2	12/02/2002		12/02/2002	

TABELA 6.18 – Controle dos objetos versionados, tabela `VOClastVersion`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	101,22,2	02/01/2002	14/03/2002	02/01/2002	
125,22,null	125,22,2	12/02/2002	17/03/2002	12/02/2002	
101,22,null	101,22,2	15/03/2002		02/01/2002	14/03/2002
101,22,null	101,22,3	15/03/2002		15/03/2002	
125,22,null	125,22,2	18/03/2002		12/02/2002	17/03/2002
125,22,null	125,22,3	18/03/2002		18/03/2002	

TABELA 6.19 – Controle dos objetos versionados, tabela `VOCversionCount`

tvoid	value	itTime	ftTime	ivtime	fvTime
101,22,null	2	02/01/2002	14/03/2002	02/01/2002	
125,22,null	2	12/02/2002	17/03/2002	12/02/2002	
101,22,null	2	15/03/2002		02/01/2002	14/03/2002
101,22,null	3	15/03/2002		15/03/2002	
125,22,null	2	18/03/2002		12/02/2002	17/03/2002
125,22,null	3	18/03/2002		18/03/2002	

6.4 Consultas

Nesta subseção são apresentadas algumas consultas sobre esses dados e seus resultados são exibidos através da ferramenta implementada. Por omissão, a data de consulta é a data de hoje (agosto de 2002) e o estado da base de dados é o atual (tempo de transação final em aberto), ou seja, dados armazenados com tempo de transação final no passado são descartados. Os resultados das consultas são apresentados na tela da interface proposta nessa dissertação.

A. Recuperar os identificadores, os títulos e os nomes das páginas de tamanho menor ou igual a 6Kb.

```
SELECT tvoid, pageTitle, nickname
FROM WebPage
WHERE pageSize <= 6;
```

Essa consulta considera apenas os dados das versões correntes, ou seja, outras versões desse tamanho não são retornadas. A consulta retorna vazio, pois as versões atuais possuem tamanhos superiores a 6Kb. Para a consulta considerar todos valores de todas as versões, deve ser alterada para:

```
SELECT tvoid, pageTitle, nickname
FROM WebPage.VERSIONS
WHERE pageSize <= 6;
```

O resultado é apresentado na figura 6.6, onde do lado esquerdo da tela aparece a consulta na sintaxe da TVQL, no lado direito a mesma consulta mapaeada para a SQL e na parte inferior o resultado da consulta. Observando o retorno da consulta, novamente, há duas informações (`PageTitle`) diferentes para o mesmo objeto ao mesmo tempo, indicando a presença de tempo ramificado para o objeto e linear para versões.

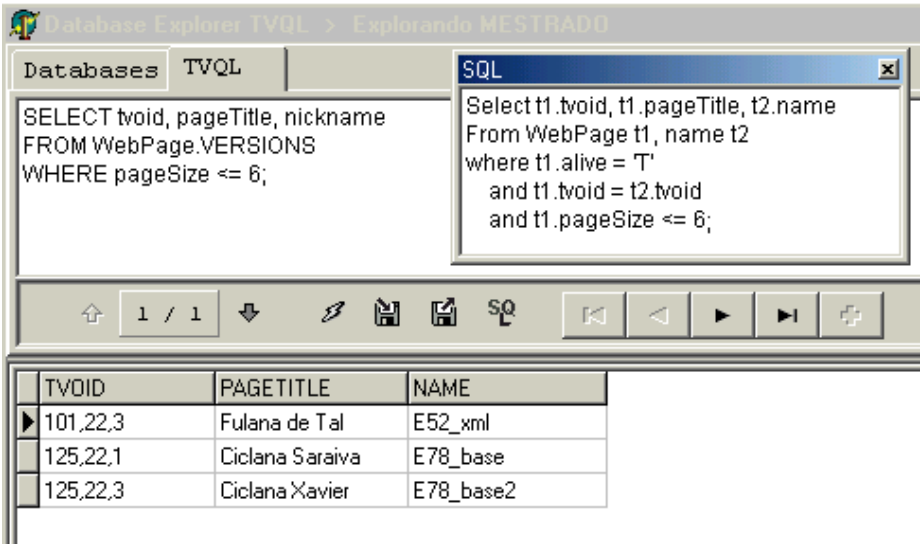

FIGURA 6.6 – Retorno da consulta A

B. Recuperar os identificadores e os títulos das versões das páginas que possuem código XML atualmente.

```
SELECT V.tvOid, V.pageTitle
FROM WebPage.versions V
WHERE V.pageText LIKE '%.xml';
```

Essa consulta considera todas as versões da classe `WebPage` (palavra `VERSIONS` na cláusula `FROM`). Se existissem versões XML em outros objetos versionados, a consulta retornaria todas elas. A figura 6.7 apresenta o resultado.

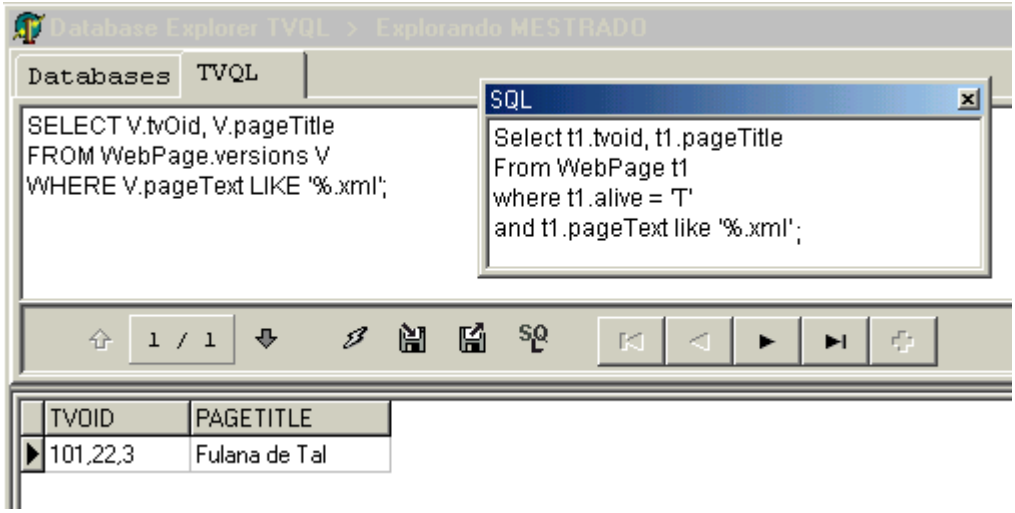

FIGURA 6.7 – Retorno da consulta B

Se a consulta considerasse apenas versões correntes (sem a palavra `.VERSIONS`), ela retornaria vazio pois os objetos versionados são `.htm`.

C. Recuperar os identificadores e os títulos, juntamente com os tamanhos e seus rótulos de validade das versões das páginas que alguma vez possuíram tamanho maior que 80Kb.

```
SELECT EVER V.tvoid, V.pageTitle, V.pageSize,
            V.pageSize.vInterval
FROM WebPage.versions V
WHERE V.pageSize > 80;
```

Essa consulta considera o histórico de todas as versões de `WebPage`. O resultado é apresentado na figura 6.8. Mesmo a cláusula `WHERE` avaliando o histórico de `pageSize`, a cláusula `SELECT` obtém o histórico de `pageTitle` também (palavra `EVER` na cláusula `SELECT`). Por isso, aparecem os dois títulos da versão "125,22,2" para o mesmo tamanho de página.

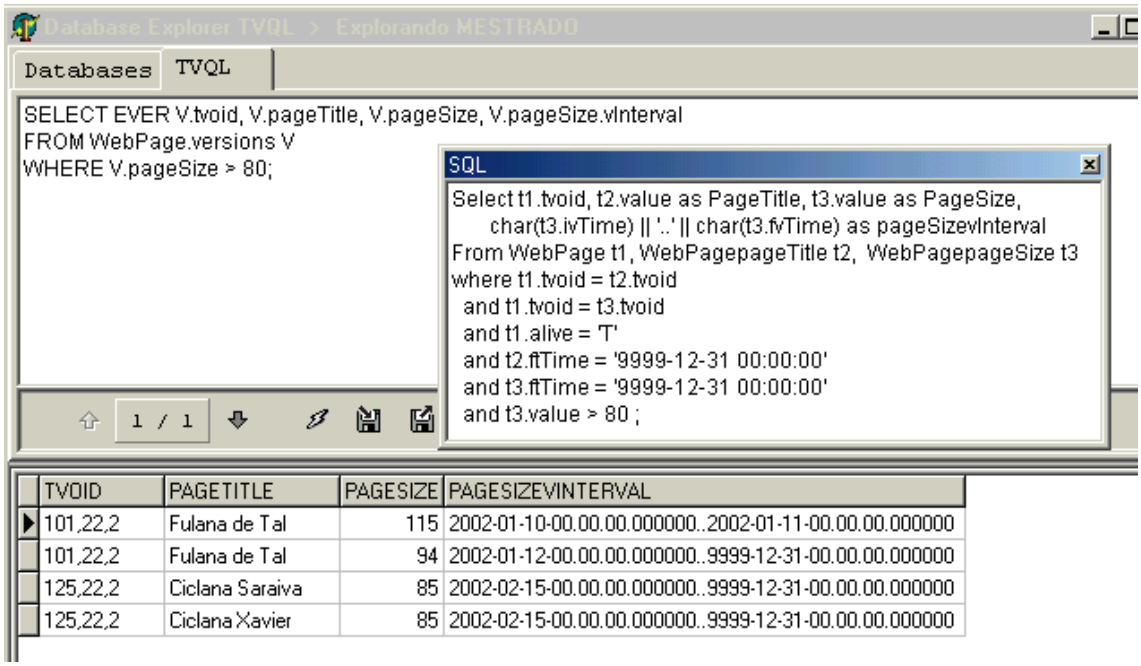

FIGURA 6.8 – Retorno da consulta C

Essa consulta considera apenas o rótulo temporal de uma propriedade. Importante notar que considerando mais de uma propriedade temporal com o devido rótulo, o resultado pode ser um produto cartesiano dos valores válidos em tempos diferentes. Para evitar o produto cartesiano, é necessário realizar uma interseção entre as validades das propriedades, como por exemplo:

```
SELECT EVER  V.tvOid, V.pageTitle,
     V.pageTitle.vInterval, V.pageSize,
     V.pageSize.vInterval
FROM WebPage.versions V
WHERE V.pageSize > 80 AND
V.pageTitle.vInterval INTERSECT V.pageSize.vInterval;
```

O resultado é apresentado na figura 6.9.

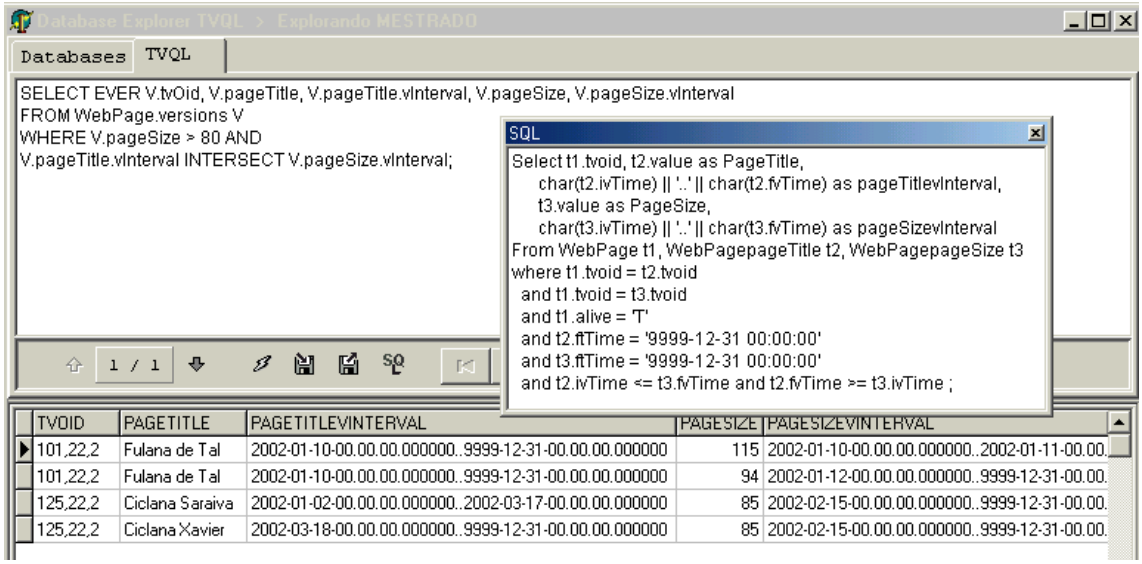

FIGURA 6.9 – Retorno da consulta C

D. Recuperar os identificadores e os tamanhos das versões das páginas das funcionárias 52 e 78 nos quais o tamanho da página da 52 é maior que da 78.

```
SELECT EVER vV52.tvoid AS E52oid,
     V52.pageSize AS E52pageSize,
     V78.tvoid AS E78oid, V78.pageSize AS E78pageSize
FROM WebPage.versions V52, WebPage.versions V78
WHERE  V52.nickname LIKE 'E52%' AND
     V78.nickname LIKE 'E78%' AND
     V52.pageSize > V78.pageSize AND
     V52.vInterval INTERSECT V78.vInterval;
```

O resultado é apresentado na figura 6.10.

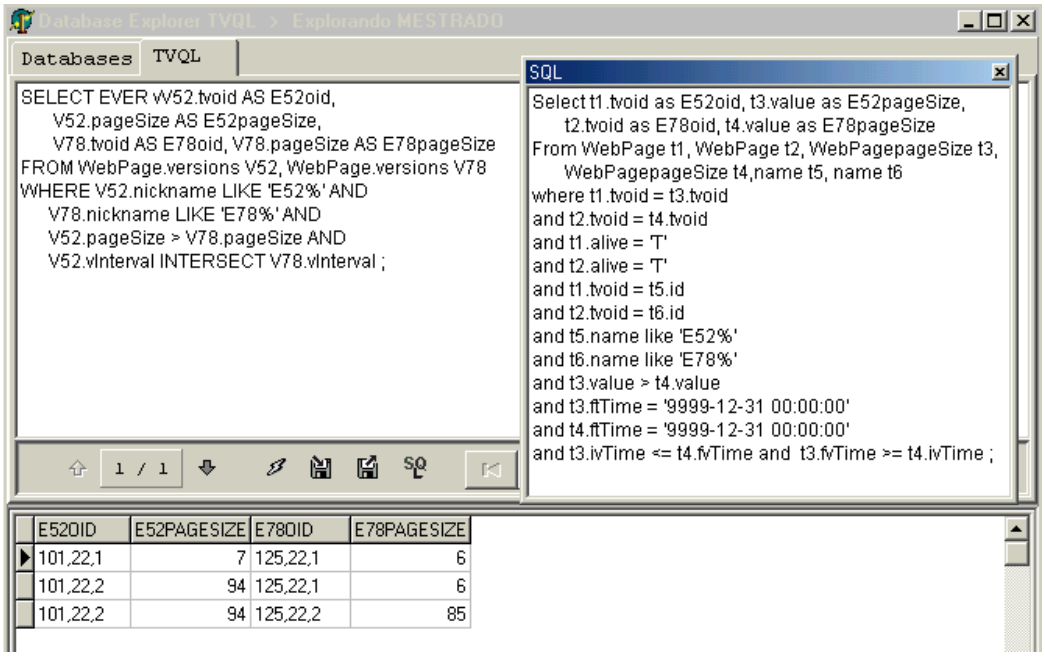

FIGURA 6.10 – Retorno da consulta D

E. Recuperar os identificadores e os títulos das versões das páginas conforme o estado da base no dia 20/02/2002.

```
SELECT EVER tvoid, pageTitle
FROM WebPage.versions
WHERE  '20/02/2002' INTO pageTitle.tInterval;
```

O resultado é apresentado na figura 6.11.

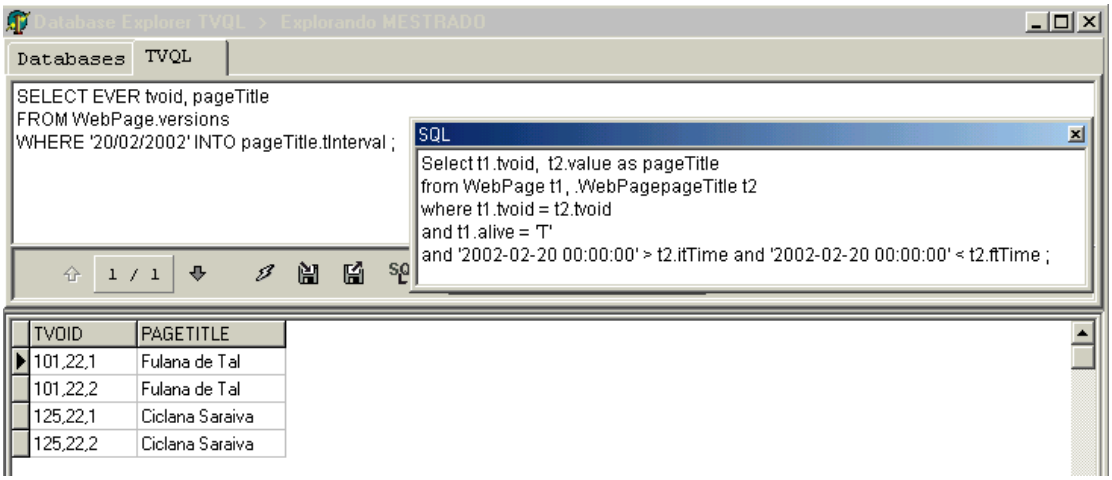

FIGURA 6.11 – Retorno da consulta E

F. Recuperar os identificadores e os títulos das versões da página da funcionária 78 durante fevereiro.

```
SELECT EVER tvoid, pageTitle
FROM WebPage.versions
WHERE  nickname LIKE 'E78%'AND
     pageTitle.vInterval INTERSECT
                    ['01/02/2002'..'28/02/2002'];
```

O mapeamento dessa consulta é apresentado na figura 6.12. O resultado não retorna "Ciclana Xavier" pois este valor começou a valer (validade inicial) em março.

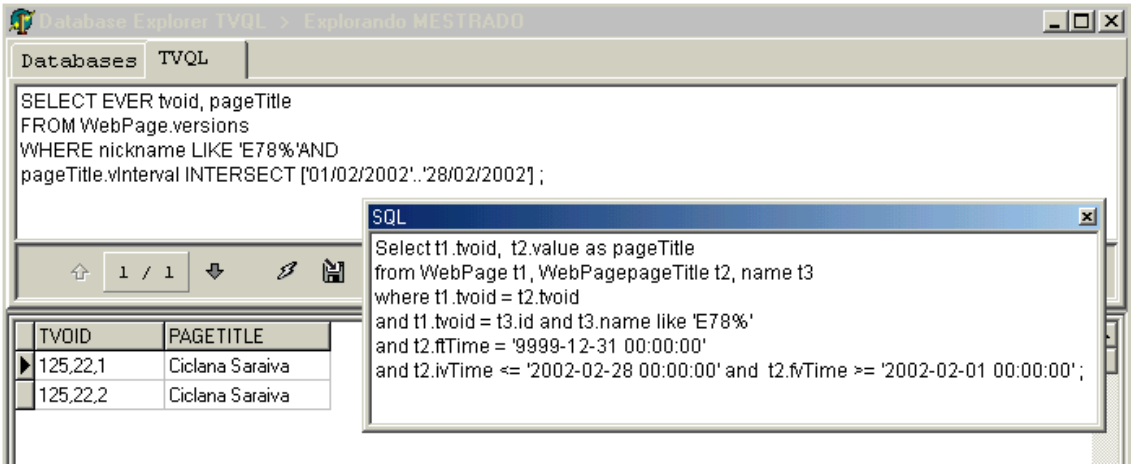

FIGURA 6.12 – Retorno da consulta F

G. Recuperar os identificadores e os títulos das primeiras versões das páginas.

```
SELECT tvoid, pageTitle
FROM WebPage.versions
WHERE  tvoid = firstVersion;
```

A figura 6.13 apresenta o resultado dessa consulta, a qual mostra como o acesso direto ao controle do objeto versionado fica transparente ao usuário que usa apenas firstVersion para acessar a primeira versão do objeto. Nesse caso, para realizar o mapeamento da consulta é preciso referenciar o controle do objeto versionado (`t1.refVOC = t2.tvoid`) e requisitar a primeira versão (`t1.tvoid = t2.firstVersion`).

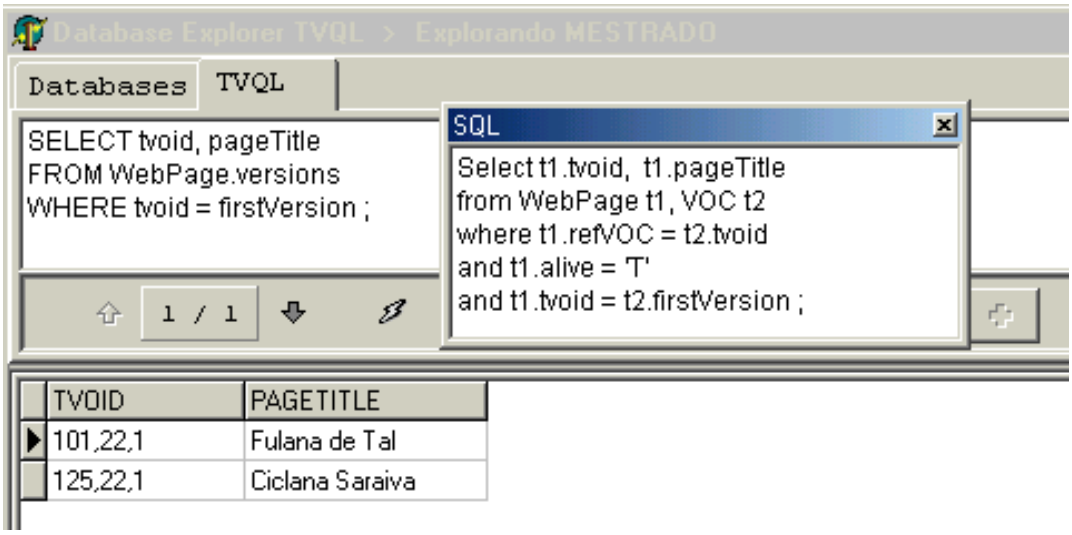

FIGURA 6.13 – Retorno da consulta G

H. Recuperar os identificadores e os títulos das versões em trabalho das páginas da base.

```
SELECT V.tvoid, V.pageTitle
FROM WebPage.versions V
WHERE V.isWorking;
```

O resultado é apresentado na figura 6.14. O resultado não retorna o valor "Ciclana Saraiva" para a versão "125,22,2", pois este valor pertence ao passado no histórico de `pageTitle`.

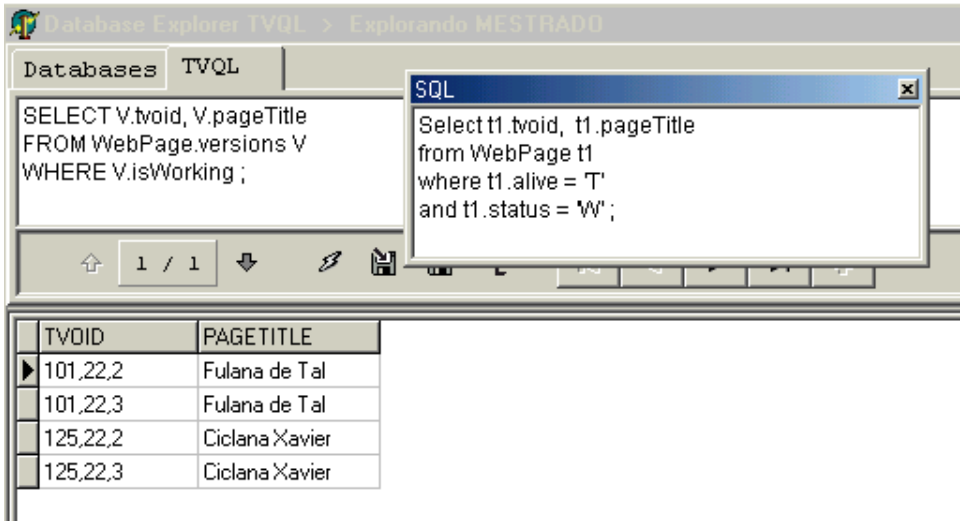

FIGURA 6.14 – Resultado da consulta H

I. Recuperar os identificadores e os títulos das versões que estão em trabalho no dia 13/02/2002.

```
SELECT EVER V.tvoid, V.pageTitle
FROM WebPage.versions V
WHERE V.isWorkingAt ('13/02/2002');
```

O resultado é apresentado na figura 6.15. O resultado não retorna as terceiras versões dos objetos, pois foram criadas em março.

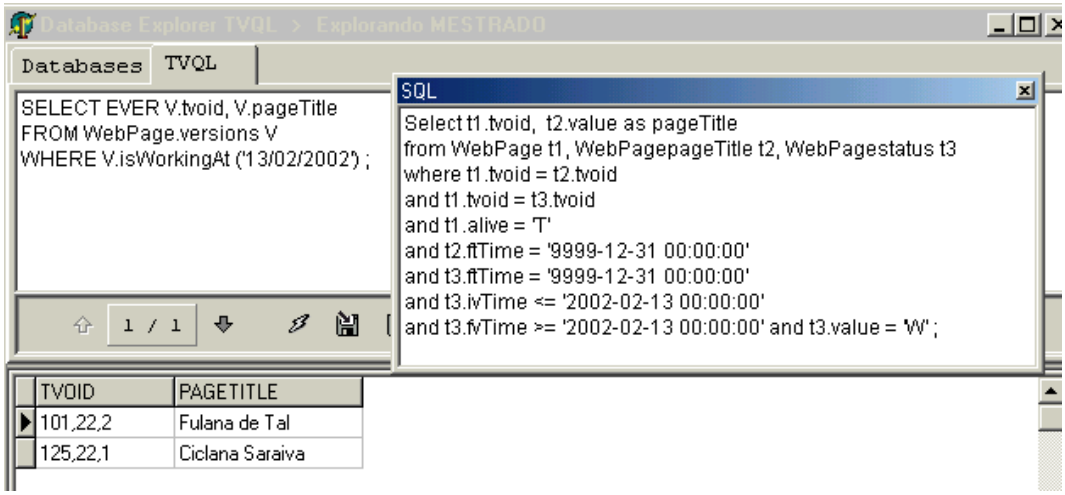

FIGURA 6.15 – Resultado da consulta I

J. Recuperar os identificadores, os históricos do status e seus rótulos das versões que são estáveis.

```
SELECT EVER tvoid, status, status.tInterval,
       status.vInterval
FROM WebPage.versions V
WHERE PRESENT (V.isStable);
```

O resultado é apresentado na figura 6.16.

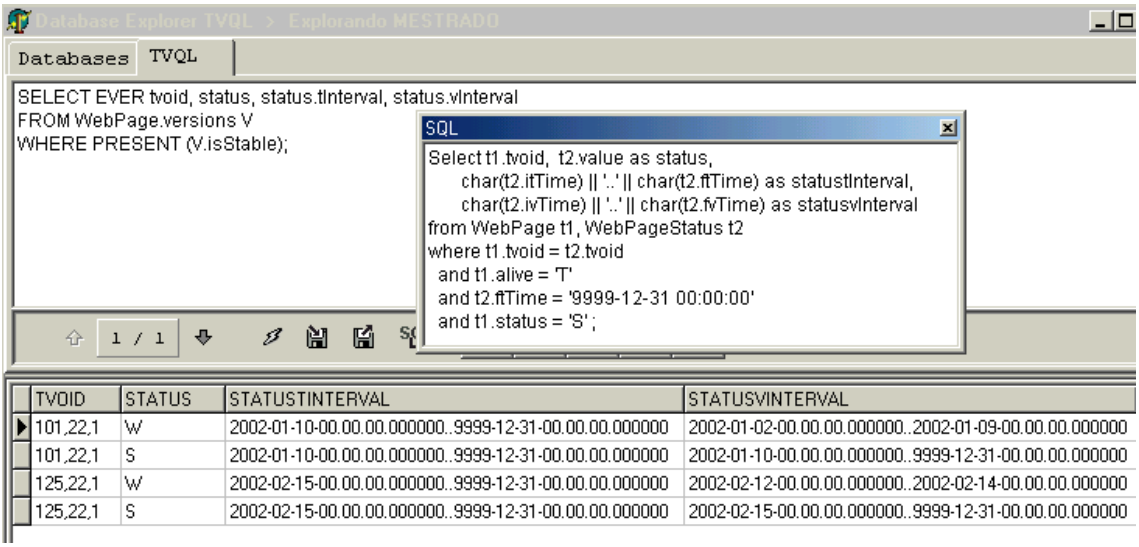

FIGURA 6.16 – Retorno da consulta J

K. Recuperar o histórico dos dados das versões de WebPage, cujo título algum dia foi "Fulana de Tal".

```
SELECT EVER tvoid, pageTitle, pageText, pageSize
FROM WebPage.versions
WHERE pageTitle.value = 'Fulana de Tal' AND
      pageTitle.vInterval INTERSECT pageText.vInterval AND
      pageTitle.vInterval INTERSECT pageSize.vInterval AND
      pageText.vInterval INTERSECT pageSize.vInterval;
```

O resultado é apresentado na figura 6.17. O resultado mostra todas as instâncias das versões da página com todos os valores dos históricos dos dados (válidos ao mesmo tempo). O produto cartesiano foi evitado com o operador `INTERSECT`.

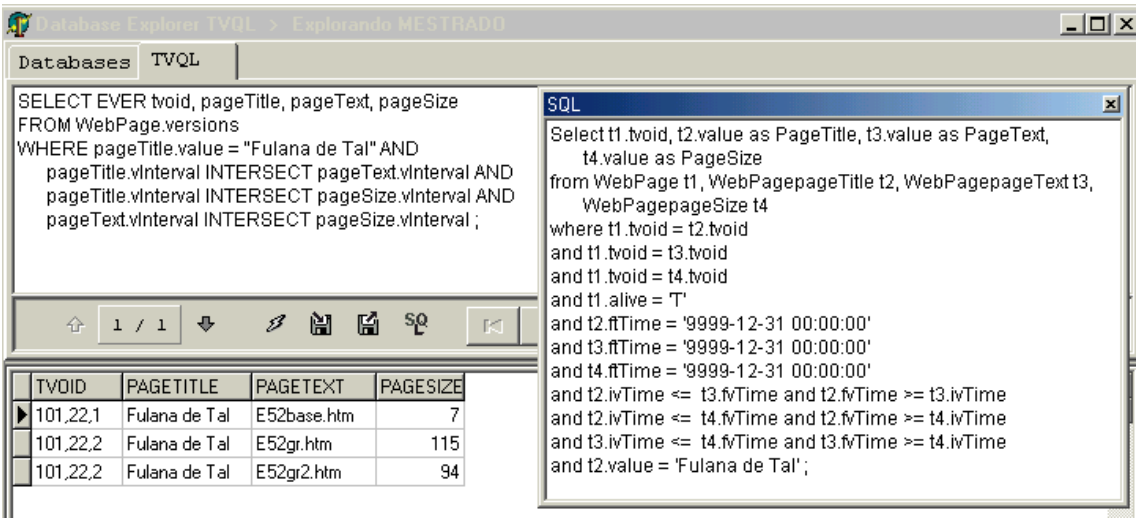

FIGURA 6.17 – Retorno da consulta K

6.5 Considerações Finais

Nesse capítulo foi apresentada a implementação do mapeamento da TVQL para SQL, através de um estudo de caso e de exemplos de consultas demonstradas e executadas em um protótipo que testa o desempenho do mapeamento definido no capítulo 5 desse trabalho.

7 Conclusões

O Modelo Temporal de Versões considera o gerenciamento das versões dos objetos em nível da aplicação, suportando a representação das informações dependentes de tempo e seqüenciamento, conforme definido pelo usuário da aplicação.

A recuperação de informações armazenadas em um banco de dados é fundamental. Para isso, os modelos de dados devem apresentar uma linguagem de recuperação de informações associada. Nesse contexto, a partir da definição do TVM, surgiu a necessidade de realizar consultas sobre os dados do modelo. Então foi definida a TVQL (*Temporal Versioned Query Language*), uma linguagem para consultas às versões considerando também o aspecto temporal do TVM.

Como não existe um SGBD para o TVM, a solução é realizar o mapeamento do modelo para um BD convencional. Depois do modelo mapeado há a necessidade de realizar consultas sobre o TVM. Então, além de mapear o modelo, a linguagem de consulta também deve ser mapeada.

Esse trabalho apresentou o mapeamento das classes do modelo e de aplicação do TVM, bem como da linguagem de consulta definida para o modelo. Um protótipo foi implementado com o objetivo de testar a funcionalidade da linguagem de consulta e do mapeamento para o DB2.

7.1 Contribuições e Limitações

As principais contribuições desse trabalho são:

- o mapeamento das classes da hieraquia do modelo TVM e das classes de aplicacação para um banco de dados relacional, pois sem isso não seria possível mapear a linguagem de consulta;
- o mapeamento da TVQL para SQL [MOR 2002];
- a implementação de um protótipo com o intuito de facilitar a recuperação dos dados armazenados, e de testar o funcionamento da linguagem de consulta e do mapeamento.

As principais limitações deste trabalho foram ditadas pela implementação do TVM no DB2, pois esse banco de dados não possui a definição de tipos lógicos e coleções. Além disso, no mapeamento das classes do modelo e de aplicação, os métodos não foram traduzidos para *triggers* e *stored procedures*, pois como o objetivo principal é consultar os dados, ou seja, mapear a TVQL, então somente classes, atributos e relacionamentos foram mapeados.

7.2 Trabalhos Futuros

Como alternativas para trabalhos futuros estão:

- a definição da álgebra relacional completa da TVQL, formalizando assim a linguagem e o mapeamento;

- a realização do mapeamento do TVM e da TVQL para um banco de dados orientado a objetos;
- a análise comparativa do desempenho das consultas em um banco de dados normal (sem as características do TVM), com as consultas em um BD que apresente as características do TVM, por exemplo, através do mapeamento realizado neste trabalho;
- o mapeamento dos métodos das classes do modelo e de aplicação do TVM para o DB2;
- implementação de uma interface visual da linguagem de consulta, assim o usuário não precisa ter conhecimento detalhado da sintaxe da TVQL.

Anexo 1 BNF da TVQL

Esta BNF é apresentada, sem recursão à esquerda, conforme a seguinte notação:

- entre [ ] para itens opcionais;
- entre { } para itens repetitivos (zero ou mais vezes);
- os símbolos terminais e não-terminais são diferenciados pela apresentação em negrito dos terminais.

```
query             ::=  SELECT [ EVER ] [ DISTINCT ] targetClause
                       FROM identificClause
                       [ WHERE [ EVER ] searchClause ]
                       [ GROUP BY groupClause { , groupClause }
                         [ HAVING logicalExpression ] ]
                       [ ORDER BY orderClause { , orderClause } ]
                       [ setOp query ] ;
targetClause      ::=  targetItem [ AS identifier ]
                          { , targetItem [ AS identifier ] }
                       | * [ , tempLabels ]
   tempLabels     ::=  intervals | instants
targetItem        ::=  oid
                       | propertyName
                       | aggrFunctions
                       | preDefInterval
                       | preDefInstant
                       | preDefLifeTime
identificClause   ::=  identificItem [ alias ] { , identificItem [ alias ] }
identificItem     ::=  className [.VERSIONS ]
searchClause      ::=  logicalExpr
groupClause       ::=  propertyName
                       | preDefInterval
                       | preDefInstant
                       | preDefLifeTime
orderClause       ::=  groupClause [ ASC | DESC ]
setOp             ::=  UNION | INTERSECTION | DIFFERENCE
propertyList      ::=  propertyName { , propertyName }
propertyName      ::=  [head.]property
        head      ::=  className [.versions ]
                       | alias
        property  ::=  identifier | preDefVProperties
preDefVProperties::=  configurationCount
                       | currentVersion
                       | firstVersion
                       | lastVersion
                       | nextVersionNumber
                       | userCurrentFlag
                       | versionCount
                       | nickname
                       | entity
aggrFunctions     ::=  sum ( propertyName )
```

```
                    | avg ( propertyName )
                    | min ( propertyName )
                    | max ( propertyName )
                    | count ( countV )
countV          ::= propertyName | *
preDefInterval  ::= [propertyName.] intervals
intervals       ::=  tInterval | vInterval
preDefInstant   ::= [propertyName.] instants
instants        ::=  tiInstant | tfInstant | viInstant | vfInstant
preDefLifeTime  ::= [head.] lifeTimes
lifetimes       ::= iLifetime | fLifetime
tvObject        ::= alias | tvObjectId
    tvObjectId  ::= " integer , integer , integer "
className       ::= identifier
alias           ::= identifier
logicalExpr     ::= logicalTerm LO
          LO    ::= { or logicalExpr LO }
logicalTerm     ::= logicalFactorLT
          LT    ::= { and  logicalTerm LT }
logicalFactor   ::= [ not ] logicalElement
logicalElement  ::= ( logicalExpr )
                    | PRESENT ( logicalExpr )
                    | tInstantExpr
                    | tIntervalExpr
                    | stateFunctions
                    | versionFunctions
                    | propertyName [ [ NOT ] LIKE pattern ]
                    | propertyName IN ( inSets )
                    | propertyName BETWEEN value AND value
                    | arithmeticExpr comparisonOp arithmeticExpr
                    | false
                    | true
tInstantExpr    ::= tInstantTerm comparisonOp tInstantTerm
tIntervalExpr   ::= tIntervalTerm intervalOp tIntervalTerm
                    | mixTerm mixOp tIntervalTerm
tInstantTerm    ::= propertyName
                    | instant
                    | preDefInstant
                    | preDefLifeTime
tIntervalTerm   ::= preDefInterval
                    | definedInterval
comparisonOp    ::= > | < | >= | <= | = | <>
intervalOp      ::= INTERSECT | OVERLAP | EQUAL
mixTerm         ::= tIntervalTerm
                    | tInstantTerm
mixOp           ::= AFTER | INTO | BEFORE
instant         ::= tempValue | now
tempValue       ::= '  dateValue [ , hourValue ] '
definedInterval ::= [ iInstant .. [ fInstant ] ]
                    | [ .. fInstant ]
iInstant        ::= instant | propertyName
fInstant        ::= instant | propertyName
stateFunctions  ::= [ head.] sFunctions
```

```
    sFunctions   ::=  isWorking
                      | isWorkingAt ( instantTerm )
                      | isStable
                      | isStableAt ( instantTerm )
                      | isConsolidated
                      | isConsolidatedAt ( instantTerm )
                      | isDeactivated
                      | isDeactivatedAt ( instantTerm )
versionFunctions::=   [ head.] vFunctions
     vFunctions ::=   isFirst
                      | isFirstAt ( instantTerm )
                      | isLast
                      | isLastAt ( instantTerm )
                      | isLessRecent
                      | isMoreRecent
                      | isSuccessorOf ( tvObjectPredecessor )
                      | isSuccessorOfAt( tvObjectPredecessor, instantTerm )
                      | isPredecessorOf( tvObjectSuccessor )
                      | isAscendantOf ( tvObjectDescendant )
                      | isDescendantOf ( tvObjectAscendant )
                      | isCurrent
                      | isCurrentAt ( instantTerm )
                      | isUserCurrent
                      | isUserCurrentAt ( instantTerm )
                      | isConfiguration
tvObjectSuccessor     ::= tvObject
tvObjectPredecessor   ::= tvObject
tvObjectAscendant     ::= tvObject
tvObjectDescendant    ::= tvObject
instantTerm           ::= instant
pattern          ::=  ' patternValue '
patternValue     ::=  { anyChar } PV
           PV    ::=  { { _ } PV1 PV }
           PV1   ::=  [ % ] { patternValue }
inSets           ::=  SELECT [ TEMPORAL ] [ DISTINCT ] propertyName
                        FROM identificClause
                        WHERE searchClause
                      | value { , value }
arithmeticExpr   ::=  term AE
                      | [ - ] arithmeticExpr
           AE    ::=  { sig term AE }
          sig    ::=  + | -
term             ::=  element TE
           TE    ::=  { op element TE }
           op    ::=  * | /
element          ::=  propertyName
                      | value
                      | ( arithmetichExpr )
dateValue        ::=  number / number / number [ number]
hourValue        ::=  number : number
value            ::=  integer | real | string | char
                      | tempValue | now | null
integer          ::=  digit { digit }
```

```
real            ::=  integer [ , integer ]
string          ::=  " { anyChar } "
anyChar         ::=  any character including blanck
digit           ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
number          ::=  digit digit
identifier      ::=  letter { ID } | _  { ID }
          ID    ::=  letter | digit | _
letter          ::=  A | B | C | D | E | F | G | H | I | J | K | L | M
                     | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
                     | a | b | c | d | e | f | g | h | i | j | k | l | m
                     | n | o | p | q | r | s | t | u | v | w | x | y | z
char            ::= ` anyChar '
```

Anexo 2 *Script* das classes do TVM

Este anexo apresenta o *script* gerado para a criação da hierarquia de classes base do TVM no DB2, usando as características relacionais do banco de dados.

```
CREATE TABLE VOC (
    tvoid               VARCHAR (20) NOT NULL,
    configCount         INTEGER,
    currentVersion      VARCHAR (20),
    firstVersion        VARCHAR (20),
    lastVersion         VARCHAR (20),
    nextVersionNumber   INTEGER,
    userCurrentF        CHAR (1) CONSTRAINT cuserCurrentF CHECK
                                 (value IN ('T','F')),
    versionCount        INTEGER,
   CONSTRAINT VOCPK PRIMARY KEY (tvoid)
);

CREATE TABLE VOCconfigCount (
        tvoid     VARCHAR(20) NOT NULL,
        value     INTEGER,
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOCConfigCountPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
);

CREATE TABLE VOCcurrentVersion (
        tvoid     VARCHAR(20) NOT NULL,
        value     VARCHAR(20),
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOCCurrentVPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
);

CREATE TABLE VOCfirstVersion (
        tvoid     VARCHAR(20) NOT NULL,
        value     VARCHAR(20),
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOCFirstVersionPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
```

```
);

CREATE TABLE VOClastVersion (
        tvoid     VARCHAR(20) NOT NULL,
        value     VARCHAR(20),
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOClastVersionPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
);

CREATE TABLE VOCuserCurrentF (
        tvoid     VARCHAR(20) NOT NULL,
        value     CHAR (1) CONSTRAINT cvalue CHECK
                           (value IN ('T','F')),
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOCuserCurrentFPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
);

CREATE TABLE VOCversionCount (
        tvoid     VARCHAR(20) NOT NULL,
        value     INTEGER,
        itTime    TIMESTAMP NOT NULL,
        ftTime    TIMESTAMP,
        ivTime    TIMESTAMP NOT NULL,
        fvTime    TIMESTAMP,
   CONSTRAINT VOCversionCountPK PRIMARY KEY (tvoid, itTime, ivTime),
   FOREIGN KEY tvoid REFERENCES VOC
);

CREATE TABLE PredSucc (
        seqPS        INTEGER NOT NULL,
        Predecessor  VARCHAR(20),
        Successor    VARCHAR(20),
        itTime       TIMESTAMP,
        ftTime       TIMESTAMP,
        ivTime       TIMESTAMP,
        fvTime       TIMESTAMP,
   CONSTRAINT PredSuccPK PRIMARY KEY (seqPS)
);

CREATE TABLE AscDesc (
        seqAD        INTEGER NOT NULL,
        Ascendant    VARCHAR(20),
        Descendant   VARCHAR(20),
        itTime       TIMESTAMP,
```

```
        ftTime       TIMESTAMP,
        ivTime       TIMESTAMP,
        fvTime       TIMESTAMP,
   CONSTRAINT AscDescPK PRIMARY KEY (seqAD)
);
```

Anexo 3 *Script* do Exemplo Ilustrativo

Este *script* corresponde à criação das tabelas provenientes do mapeamento das classes de aplicação do TVM, que nesse caso são de um sistema para criação de *websites*, conforme apresentado na seção 6.3 desse trabalho.

```
CREATE TABLE Font (
       tvoid      VARCHAR(20) NOT NULL,
       fontColor  VARCHAR(100),
       fontEffect VARCHAR(100),
       fontSize   INTEGER,
       fontType   VARCHAR(100),
     CONSTRAINT FontPK PRIMARY KEY (tvoid)
);

CREATE TABLE PagePattern (
       tvoid            VARCHAR(20) NOT NULL,
       refVOC           VARCHAR(20),
       alive            CHAR(1) CONSTRAINT calive CHECK
                                (value IN ('T','F')),
       configuration    CHAR(1) CONSTRAINT cconfiguration CHECK
                                (value IN ('T','F')),
       status           CHAR(1) CONSTRAINT cstatus CHECK
                                (value IN ('W','S','C','D')),
       backgrdColor     VARCHAR(100),
       backgrdImage     VARCHAR(100),
       banner           VARCHAR(100),
       associatedWith   VARCHAR(100),
       default          VARCHAR(50),
     CONSTRAINT PagePatternPK PRIMARY KEY (tvoid),
   FOREIGN KEY default   REFERENCES Font(tvoid),
   FOREIGN KEY associatedWith   REFERENCES WebSite(tvoid)
);

CREATE TABLE PagePatternalive (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK
                            (value IN ('T','F')),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT PagePatternalivePK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES PagePattern(tvoid)
);

CREATE TABLE PagePatternstatus (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK
```

```
                            (value IN ('W','S','C','D')),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT PagePatternstatuPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES PagePattern(tvoid)
);

CREATE TABLE PagePatternbanner (
       tvoid        VARCHAR(20) NOT NULL,
       value        VARCHAR(100),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT PagePatternbannePK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES PagePattern(tvoid)
);

CREATE TABLE WebPage (
       tvoid            VARCHAR(20) NOT NULL,
       refVOC           VARCHAR(20),
       alive            CHAR(1) CONSTRAINT calive CHECK
                               (value IN ('T','F')),
       configuration    CHAR(1) CONSTRAINT cconfiguration CHECK
                               (value IN ('T','F')),
       status           CHAR(1) CONSTRAINT cstatus CHECK,
                               (value IN ('W','S','C','D')),
       pageSize         INTEGER,
       pageText         VARCHAR(100),
       pageTitle        VARCHAR(100),
     CONSTRAINT WebPagePK PRIMARY KEY (tvoid)
);

CREATE TABLE WebPagealive (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK
                            (value IN ('T','F')),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebPagealivePK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebPage(tvoid)
);

CREATE TABLE WebPagestatus (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK
                            (value IN ('W','S','C','D')),
       itTime       timestamp NOT NULL,
```

```
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebPagestatusPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebPage(tvoid)
);

CREATE TABLE WebPagepageSize (
       tvoid        VARCHAR(20) NOT NULL,
       value        INTEGER,
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebPagepageSizePK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebPage(tvoid)
);

CREATE TABLE WebPagepageText (
       tvoid        VARCHAR(20) NOT NULL,
       value        VARCHAR(100),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebPagepagetextPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebPage(tvoid)
);

CREATE TABLE WebPagepageTitle (
       tvoid        VARCHAR(20) NOT NULL,
       value        VARCHAR(100),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebPagepageTitlPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebPage(tvoid)
);

CREATE TABLE WebSite (
       tvoid            VARCHAR(20) NOT NULL,
       refVOC           VARCHAR(20),
       alive            CHAR(1) CONSTRAINT calive CHECK
                               (value IN ('T','F')),
       configuration    CHAR(1) CONSTRAINT cconfiguration CHECK
                               (value IN ('T','F')),
       status           CHAR(1) CONSTRAINT cstatus CHECK,
                               (value IN ('W','S','C','D')),
       online           CHAR(1) CONSTRAINT conline CHECK
                               (value IN ('T','F')),
       URL              VARCHAR(100),
```

```
       pattern   VARCHAR(50),
       initialPage      VARCHAR(50),
     CONSTRAINT WebSitePK PRIMARY KEY (tvoid),
     FOREIGN KEY pattern   REFERENCES PagePattern(tvoid),
     FOREIGN KEY initialPage   REFERENCES WebPage(tvoid)
);

CREATE TABLE WebSitealive (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK,
                            (value IN ('T','F')),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebSitealivePK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebSite(tvoid)
);

CREATE TABLE WebSitestatus (
       tvoid        VARCHAR(20) NOT NULL,
       value        CHAR(1) CONSTRAINT cvalue CHECK,
                            (value IN ('W','S','C','D')),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebSitestatusPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebSite(tvoid)
);

CREATE TABLE WebSiteURL (
       tvoid        VARCHAR(20) NOT NULL,
       value        VARCHAR(100),
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebSiteURLPK PRIMARY KEY (tvoid,itTime,ivTime),
     FOREIGN KEY tvoid REFERENCES WebSite(tvoid)
);

CREATE TABLE WebSitepattern (
       tvoid1       VARCHAR(20) NOT NULL,
       tvoid2       VARCHAR(20) NOT NULL,
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
    CONSTRAINT WebSitepatternPK PRIMARY KEY
                    (tvoid1, tvoid2, itTime, ivTime),
     FOREIGN KEY tvoid1 REFERENCES WebSite(tvoid),
```

```
     FOREIGN KEY tvoid2 REFERENCES PagePattern(tvoid)
);

CREATE TABLE WebSiteinitialPage (
       tvoid1       VARCHAR(20) NOT NULL,
       tvoid2       VARCHAR(20) NOT NULL,
       itTime       timestamp NOT NULL,
       ftTime       timestamp,
       ivTime       timestamp NOT NULL,
       fvTime       timestamp,
     CONSTRAINT WebSiteinitialPPK PRIMARY KEY
                  (tvoid1, tvoid2,itTime,ivTime),
     FOREIGN KEY tvoid1 REFERENCES WebSite(tvoid),
     FOREIGN KEY tvoid2 REFERENCES WebPage(tvoid)
);
```

Bibliografia

[ABD 97] ABDESSALEM, T.; JOMIER, G. Vql: A query language for multiversion databases. In: INTERNATIONAL. WORKSHOP ON DATABASE PROGRAMMING LANGUAGES, DPBL, 6., 1997, Paris. **Database programming languages:** proceedings. Berlin: Springer-Verlag, 1998. p.161-179. (Lecture Notes in Computer Science, 1392).

[AGR 91] AGRAWAL, R.; BUROFF, S; GEHANI, N; SHASHA, D. Object versioning in Ode. In: IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING - ICDE, 7., 1991, Kobe. **Proceedings**… Los Alamitos, IEEE Computer Society, 1991. p.446-455.

[ALK 2001] AL-KHUDAIR, A.; GRAY, W.A.; MILES, J.C. Object-Oriented Versioning in a Concurrent Engineering Design Environment. In: BRITISH NATIONAL CONFERENCE ON DATABASES – BNCOD, 18., 2001, Oxford, Inglaterra. **Advances in databases:** proceedings… Berlin: Springer-Verlag, 2001.

[AND 97] ANDONOFF, E.; HUBERT, G.; LE PART, A.; ZURFLUH G. A Query Algebra for Object-Oriented Databases Integrating Versions. In: IEEE BASQUE INTERNATIONAL WORKSHOP ON INFORMATION TECHNOLOGY – BIWIT, 3., 1997. **Proceedings**… Paris: IEEE Computer Society, 1997. p. 62-72.

[AND 98] ANDONOFF, E.; HUBERT, G.; LE PART, A. A Database Interface Integrating a Querying Language for Versions. In: ADVANCES IN DATABASES AND INFORMATION SYSTEMS – ADBIS, 1998, Poznan, Poland. **Proceedings**… Dordrecth: Springer-Verlag, 1998. p. 200-211. (Lecture Notes in Computer Science, 1475).

[BEE 88] BEECH, D.; MAHBOD, B. Generalized Version Control in an Object-Oriented Database. In: IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING - ICDE, 4., 1988, Los Angeles. **Proceedings**… Los Angeles: IEEE Computer Society, 1988. p.14-22.

[BJÖ 89] BJÖRNERSTEDT, A.; HULTÉN, C. Version Control in an Object-Oriented Architecture. In: KIM, W.; LOCHOVSKY, F.H. (Ed.). **Object-Oriented Concepts, Databases, and Applications.** New York: ACM Press, 1989. p. 451-485.

[BOW 2001] BOWMAN, I.T,; TOMAN, D. Optiming temporal queries: efficient handling of duplicates. In: INTERNATIONAL. SYMP. ON TEMPORAL REPRESENTATION AND REASONING - TIME, 8., 2001, Cividale del Friuli, Itália. **Proceedings**… Milan: IEEE Computer Society, 2001. p.93-100.

[CAV 95] CAVALCANTI, J. M. B et al. Uma abordagem para a implementação de um modelo temporal orientado a objetos usando SGBDs relacionais. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 10.,1995, Recife. **Anais...** Recife: UFPE, 1995.

[CEL 90] CELLARY, W.; JOMIER, G. Consistency of Versions in Object-Oriented Databases. In: CONFERENCE ON VERY LARGE DATA BASES – VLDB, 16., 1990, Brisbane. **Proceedings**… Brisbane: Springer-Verlag, 1990. p. 432-441.

[CHI 97] CHIEN, S-Y.; TSOTRAS, V.J.; ZANIOLO, C. Version Management of XML Documents. In: WORLD WIDE WEB AND DATABASES – WEBDB, 3., 2000, Dallas, EUA. Berlin: Springer-Verlag, 1997. p. 75-80.

[CHI 2001] CHIEN, S-Y.; TSOTRAS, V.J.; ZANIOLO, C. Efficient Management of Multiversion Documents by Object Referencing. In: INTERNATIONAL CONF. OF VERY LARGE DATABASES, 27., 2001, Roma, Itália. **Proceedings**… Roma: – VLDB, 2001. p. 291-300.

[CHI 2002] CHIEN, S-Y.; TSOTRAS, V.J.; ZANIOLO, C; ZHANG, D. Efficient Complex Query Support for Multiversion XML Documents. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY - EDBT, 8., 2002, Praga, República Tcheca. Berlin: Springer-Verlag, 2002. p. 161-178. (Lecture Notes in Computer Science, 2287).

[CON 98] CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. **ACM Computing Surveys**, New York, v.30, n.2, p.232-282, June 1998.

[COW 2000] COWLEY, W.; PLEXOUSAKIS, D. Temporal Integrity Constraints with Indeterminancy. In: INTL. CONF. ON VERY LARGE DATA BASES, 26., 2000, Cairo, Egito. **Proceedings**… Cairo: VLDB, 2000. p. 441-450.

[DAD 84] DADAM, P.; LUM, V.; WERNER, H. D. Integration of Time Versions into a Relational Database System. In: CONFERENCE ON VERY LARGE DATABASES – VLDB, 10., 1984, Singapore. **Proceedings…** San Mateo: Morgan Kauffman, 1984. p. 509-522

[EDE 98] EDELWEISS, N. Banco de Dados Temporais: Teoria e Prática. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA – JAI, 17.; CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, IB., 1998. **Anais...** Belo Horizonte: SBC, 1998. p 225-282.

[EDE 2000] EDELWEISS, N.; HÜBLER, P.; MORO, M.M.; DEMARTINI, G. A Temporal Database Management System Implemented on Top of a Conventional Database. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY – SCCC, 20., 2000, Santiago do Chile. **Proceedings**… Los Alamitos: IEEE Computer Society, 2000. p. 58-67.

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

[GAL 2002] GALANTE, R.; ROMA, A.; JANTSCH, A.; EDELWEISS, N.; SANTOS, C. Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Databases. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA 2002, Aix-en-Provence, France. **Database and**

Expert systems applications: proceedings… Berlin: Springer-Verlag, 2002. (Lecture Notes in Computer Science, 2453).

[GOL 95] GOLENDZINER, L. **Um Modelo de Versões para Banco de Dados Orientados a Objetos**. 1995. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[GOR 2001] GORALWALLA, I.A.; LEONTIEV, Y.; OZSU, M.T., SZAFRON, D.; COMBI, C. Temporal Granularity: Completing the Puzzle. **Journal of Intelligent Information Systems**, Norwell, v. 16, n. 1, p.41-63, 2001.

[HEL 75] HELD, G. D.; STONEBRAKER, M.; WONG, E. INGRESS - A relational data base management system. In: AFIPS NATIONAL COMPUTER CONFERENCE. 1975. CA, Anaheim. **Proceedings**… Montvale: AFIPS, 1975. p.409-416.

[HÜB 2000] HUBLER, P. N. **Definição de um Gerenciador para o Modelo de Dados TF-ORM**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[JEN 98] 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-Verlag, 1998. p. 367-405.

[JEN 99] JENSEN, C.S. **Temporal Database Management**. Aalborg: Department of Computer Science, Aalborg University, 1999. Dr. technical thesis. Disponível em: <http://www.cs.auc.dk/~csj/Thesis/>. Acesso em: 12 ago. 2001.

[KAT 90] KATZ, R. H. Toward a unified framework for version modeling in engineering databases. **ACM Computing Surveys**, New York, v22. n. 1990, p. 375-408, 1990.

[KIM 89] KIM, W.; BERTINO, E.; GARZA, J. F. Composite objects revisted. In: ACM SIGMOND INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1989, Oregon. **Proceedings**… New York: ACM Press, 1989. p.337-347.

[LAU 97] LAUTEMANN, S. Schema Versions in Object-Oriented Database Systems. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATION, 5., 1997, Melbourne. **Proceedings**... Singapore: World Scientific Press, 1997. p.323-332.

[LIW 2001] LI, W.; SNODGRASS, R.T.; DENG, S.; GATTU, V.K.; KASTHURIRANGAN, A. Efficient Sequenced Temporal Integrity Checking In: IEEE INTL. CONF. ON DATA ENGINEERING, ICDE, 2001, Heidelburg. **Proceedings**… Heidelburg: IEEE Computer Society, 2001. p. 131-140.

[MEL 94] MELLO, R.S.; GOLENDZINER, L.G. Uma Linguagem visual de consulta para o ambiente STAR. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 9., 1994, São Carlos. **Anais**... São Carlos: USP, 1994. p. 260-275.

[MIR 2000] MIRBEL, I,; PERNICI, B.; SELLIS, T.K.; TSERKEZOGLOU, S.; VAZIRGIANNIS, M. Checking the Temporal Integrity of Interactive

Multimedia Documents. **VLDB Journal**, Heidelberg: Springer-Verlag, v. 9, n. 2, p. 111-130, 2000.

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

[MOR 2001b] 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. (RP-308).

[MOR 2002] MORO, M. M.; ZAUPA, A. P.; EDELWEISS, N.; SANTOS, C. S. TVQL – Temporal Query Language. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA 2002, Aix-en-Provence, France. **Proceedings…** Berlin: Springer-Verlag, 2002 (Lecture Notes in Computer Science, 2453).

[MOV 99] MOREIRA, Viviane Pereira; EDELWEISS, N. **Queries to Temporal Databases Supporting Schema Versioning**. STAR. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 9., 1999, Florianópolis. **Anais**... Florianópolis: UFSC, 1999.

[NOR 98] NORONHA, M.A.; GOLENDZINER, L.G.; SANTOS, C.S. dos. Extending a Structured Document Model with Version Control. In: INTERNATIONTAL DATABASE ENGINEERING & APPLICATIONS SYMPOSIUM, 1998, Cardiff, Inglaterra. **Proceedings**... Cardiff, UK: IEEE Computer Society, 1998. p. 234-242.

[OLI 2001] OLIBONI, B.; QUINTARELLI, E.; TANCA, L. Temporal aspects of semistructured data. In: Intl. SYMP. ON TEMPORAL REPRESENTATION AND REASONING - TIME, 8., 2001, Cividale del Friuli, Itália. **Proceedings**… Cividale del Friuli: IEEE Computer Society Press, 2001. p.119-127.

[ROD 2000] RODDICK, J.F.; SCHREFK, M. Towards an Accommodation of Delay in Temporal Active Datases. In: AUSTRALASIAN DATABASE CONFERENCE, 11., 2000, Canberra, Austrália. **Proceedings**… Canberra: IEEE Computer Science Press, 2000. p. 115-119.

[ROL 99] RODRÍGUEZ, L; OGATA, H.; YANO, Y. TVOO: A Temporal Versioned Object-Oriented data model. **Information Sciences**, [S.1.], v.114, n.1-4, p. 281-300, 1999.

[ROS 93] ROSE, E.; SEGEV, A. TOOSQL – A Temporal Object-Oriented Query Language. In: INTERNATIONAL CONFERENCE ON ER-APPROACH, 12., 1993, Arlington. **Proceedings**… Dordrecth: Springer-Verlag, 1993. p. 122-136. (Lecture Notes in Computer Science, 823).

[SAR 90] SARDA, N. L. Extensions to SQL for Historical Databases. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 2, n. 2, p. 220-230, June 1990.

[SIM 98] SIMONETTO, Eugênio de Oliveira. **Uma Proposta para a Incorporação de Aspectos Temporais, no Projeto Lógico de Bancos**

de Dados, em SGBDs Relacionais. 1998. 73p. Dissertação (Mestrado em Ciência da Computação) - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

[SNO 87] SNODGRASS, R. T. The temporal query language TQuel. **ACM Transacations on Database Systems**, New York, v.12, n.2, p. 247-298, 1987.

[SNO 95] SNODGRASS, R. **The TSQL2 Temporal Query Language**. Boston: Kluwer Academic Publishers, 1995. 704 p.

[SNO 96] SNODGRASS, R.; BÖHLEN, M.; JENSEN, C.S.; STEINER, A. **Adding Valid Time to SQL/Temporal**. ANSI X3H2-96-501r2, ISO. October 1996. Disponível em: <ftp://ftp.cs.arizona.edu/tsql/tsql2/sql3/ansi-96-501.pdf>. Acesso em: 03 ago. 2001.

[SNO 98] SNODGRASS, R.; BÖHLEN, M.; JENSEN, C.; STEINER, A. Transitioning Temporal Support in TSQL2 to SQL3. In: ETZION, O.; JAJODIA, S.; SRIPADA, S. (Ed.). **Temporal Databases Research and Practice**. Berlin: Springer-Verlag, 1998. p. 150-194.

[SNO 2000] SNODGRASS, R.T. **Developing Time-Oriented Database Applications in SQL**. San Francisco: Morgan Kaufmann, 2000.

[SOA 95] SOARES, L.; RODRIGUEZ, N.; CASANOVA, M. Nested composite nodes and version control in an open hypermedia system. **Information Systems**, Exeter, v.20, n.6, p.501-591, 1995.

[TAN 93] TANSEL, C. G. et. al. (Ed.). **Temporal Databases – Theory, Design and Implementation**. Redwood City: The Benjamin/Cummings, 1993.

[VAI 2001] VAISMAN, A.; MENDELZON, A.O. A Temporal Query Language for OLAP: Implementation and a Case Study. In: BIENNAL WORKSHOP ON DATABASES AND PROGRAMMING LANGUAGES – DBPL, 2001, Roma, Itália. **Proceedings**… Disponível em: <http://www.cs. toronto.edu/~mendel/ dwbib.html>. Acesso em: 14 jan. 2002.

[WAG 91] WAGNER, F.R.; LIMA, A.H.V.; GOLENDZINER, L.G.; IOCHPE, C. STAR – Um Ambiente para a Integração de Ferramentas de Projeto de Sistemas Digitais. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 6., 1991. **Anais**... Belo Horizonte, 1991. p. 176-185.

[WUU 93] WUU, G. T. J; DAYAL, U. A Uniform Model for Temporal and Versioned Object-Oriented Databases. In: TANSEL, A. et al. (Ed.). **Temporal Databases:** Theory, Design, and Implementation. Redwood City: The Benjamin/Cummings, 1993. chap. 10, p. 230-247.

[ZAN 97] ZANIOLO, C. et al. **Advanced Database Systems**. San Francisco: Morgan Kaufmann, 1997. p.574.