Implementação de um Banco de Dados Temporal Utilizando um SGBD Convencional

Patrícia Nogueira Hübler *
Nina Edelweiss *
Tanisi Pereira de Carvalho **

* Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre - RS - Brasil
{nina / hubler} @ inf.ufrgs.br

** Universidade Luterana do Brasil
Centro de Ciências Naturais e Exatas
Curso de Informática
Gravataí - RS - Brasil

Resumo

Embora as pesquisas em bancos de dados temporais se estendam já por mais de 20 anos, existem atualmente muito poucos sistemas implementados. Muitos modelos temporais foram propostos, estendendo os modelos de dados tradicionais de modo a capturar também os aspectos temporais. Em vez de desenvolver SGBDSs específicos para estes modelos temporais, uma forma possível de implementar um banco de dados temporal é mapear o modelo temporal para um banco de dados comercial tradicional. Neste mapeamento as informações temporais, implícitas no modelo temporal, devem ser explicitamente representadas e manipuladas. Este artigo apresenta a implementação de um modelo de dados temporal sobre o banco de dados Oracle. Informações temporais são anexadas aos atributos dinâmicos. A manutenção da história das informações é controlada por gatilhos. Podem ser recuperadas informações relativas ao presente, passado ou futuro.

Abstract

Researches in the temporal database area have been developed for more than 20 years, but until now very few implemented systems are found. Several temporal models were proposed usually extending traditional data models to capture also the temporal aspects. In place of developing a DBMS for these temporal models, a temporal database can be implemented mapping the temporal data model to a non-temporal commercial database. Temporal information, implicit in the temporal data model, shall be explicitly represented and manipulated. This paper presents the implementation of a temporal data model using the Oracle database. Temporal information is added to the dynamic attributes. Triggers control the maintenance of information history. Present, past and future information can be retrieved.

1. Introdução

O aspecto temporal está presente na maior parte das aplicações do mundo real. Apresentam-se não somente como informações temporais (por ex., datas, períodos, prazos), mas também como restrições de integridade temporal que controlam a evolução de uma aplicação. Os sistemas de informação correspondentes a estas aplicações apresentam informações temporais em diferentes níveis com objetivos diferenciados, tais como: (i) informações temporais a serem armazenadas no banco de dados sob forma de atributos temporais; (ii) relacionamento temporal entre processos executados na aplicação; (iii) pré e pós-condições à execução de tarefas, representadas através de regras temporais; (iv) condições de integridade temporal para o banco de dados. Entretanto, os bancos de dados que são utilizados para armazenar as informações relativas às aplicações não apresentam suporte para o aspecto temporal.

A constatação de que as informações evoluem com a passagem do tempo levou ao desenvolvimento de bancos de dados temporais [CLI 95, EDE 94, 98, JEN 97, OZO 95, TAN 93, ZAN 97]. Para que isto seja possível, informações temporais são associadas aos dados armazenados, identificando quando a informação foi definida (tempo de transação) ou o tempo de sua validade real na aplicação (tempo de validade). Regras temporais controlam a evolução destas informações.

O grande número de modelos temporais propostos nas pesquisas desenvolvidas ao longo destes anos demonstra claramente a relevância deste tema e a necessidade da representação temporal. A maior parte dos modelos propostos são extensões de modelos tradicionais já existentes – modelos relacionais [CLI 87, GAD 88, LOR 88, NAV 89, SAR 90, SNO 87, TAN 86], orientados a objetos [EDE 93, KAF 92, ROS 91, SU 91, WUU 93], e Entidade-Relacionamento [LOU 91, ELM 93, TAU 91].

Embora as pesquisas em bancos de dados temporais datem de mais de 20 anos, poucas implementações realmente executáveis são encontradas. Entre estas podemos citar o *Temporal ODBMS TIGUCAT* [OZS 95], em desenvolvimento na Universidade de Alberta, no Canadá; e o BD Postgres, que apresenta suporte ao tempo de transação. Várias experiências de implementação têm sido desenvolvidas, sob forma de protótipos, nos quais se baseiam estudos de problemas encontrados – sobretudo problemas de armazenamento de grandes volumes de dados e de recuperação de informações.

Este artigo apresenta a implementação de um banco de dados temporal utilizando, como base, um banco de dados comercial tradicional – o BD Oracle. O modelo temporal utilizado é um modelo Entidade-Relacionamento (ER) estendido com aspectos temporais. A opção por um modelo ER deve-se ao fato de sua larga utilização na fase de modelagem conceitual de aplicações. É apresentado o mapeamento realizado entre o modelo ER temporal para o modelo relacional do BD utilizado, identificando a forma como as informações temporais são armazenadas. É feita, ainda, uma análise das possibilidades de recuperação de informações.

O artigo está organizado na forma apresentada a seguir. Na seção 2 encontra-se o estudo de caso desenvolvido para validar a implementação. A seção 3 apresenta o modelo ER temporal criado para representar as informações. Na seção 4, o mapeamento do modelo ER temporal para o banco de dados relacional Oracle é apresentado. A seção 5 contém regras para as operações de atualização e remoção em um banco de dados temporal. Na seção 6 estão as regras para manutenção das informações, bem como a utilização de *triggers*. Os

operadores temporais e suas regras de definições estão na seção 7. A seção 8 contém as visões e as *stored functions*. As consultas realizadas sobre o banco de dados temporal são apresentadas na seção 9 e, a seguir, na seção 10, encontram-se as conclusões deste artigo.

2. Estudo de caso

O estudo de caso foi realizado na Gravel Administradora de Consórcios Ltda. – Gravataí – RS, na parte forte da empresa: os consórcios. O sistema atual gerencia todas as informações referentes: (i) aos consorciados; (ii) aos grupos aos quais eles pertencem; (iii) às prestações dos bens adquiridos; (iv) às assembléias realizadas durante a vigência do grupo; (v) aos bens que compõem determinado grupo e (vi) aos vendedores que negociam tais bens.

Um dos grandes problemas da Gravel é a redundância de informações, o que gera um gasto exagerado de memória e a dificuldade de manutenção destas informações. Outra dificuldade é a manipulação de informações previamente conhecidas que apenas podem ser inseridas, no sistema, no dia da validade inicial. Um problema típico de manutenção de dados temporais.

O objetivo do estudo de caso foi determinar como as informações seriam mapeadas para a base de dados, quais dados seriam temporizados e quais seriam estáticos. Com o estudo de caso, foi possível validar a importância de um banco de dados temporal, bem como sua implementação.

A descrição completa do estudo de caso encontra-se em [HUB 97].

3. O Modelo ER Temporal Utilizado

Nesta implementação foi utilizado um modelo de dados temporal ER (Entidade-Relacionamento) genérico, com o objetivo de ser utilizado na fase de modelagem conceitual de uma aplicação. O modelo foi definido com base em uma análise de outros modelos ER temporais propostos nos últimos anos, com ênfase nos modelos ERT de Loucopoulos [LOU 91], TEER de Elmasri, Wuu e Kouramajian [ELM 93] and TER de Tauzovich [Tauzovich 91].

O tempo é modelado como sendo discreto e linearmente ordenado. Três diferentes primitivas temporais podem ser utilizadas para representar a informação temporal: pontos no tempo, intervalos temporais ou elementos temporais (conjuntos disjuntos de intervalos temporais). Os pontos no tempo podem ser utilizados para representar o início da validade de uma informação, quando uma determinada variação é considerada. Intervalos temporais representam o período completo no qual uma informação possui o mesmo valor. E elementos temporais permitem "reincarnação" de elementos – o mesmo valor pode ser válido durante dois ou mais intervalos temporais diferentes. A escolha da primitiva temporal é a base do modelo de dados, e define como a informação pode ser representada e o que a linguagem de consulta correspondente poderá recuperar. Intervalos temporais foram utilizados no modelo ER temporal deste artigo.

Nem todas as informações variam com o passar do tempo. Para simplificar a implementação, atributos estáticos e dinâmicos devem ser identificados. Atributos estáticos não necessitam de informações temporais, já que o seu valor não muda com o passar do tempo. Para os atributos dinâmicos – aqueles cujo valor muda durante a evolução da aplicação – são acrescentados rótulos temporais. Todos os valores definidos serão conservados no banco

de dados, com o rótulo correspondente, informando o período no qual o valor é válido. Duas informações diferentes podem ser utilizadas como rótulos temporais – o *tempo de transação*, correspondendo ao tempo no qual o valor é definido no banco de dados, e o *tempo de validade*, representando à validade da informação no mundo real. Esta escolha define se o banco de dados correspondente deverá ser um *Banco de Dados de Tempo de Transação,* um *Banco de Dados de Tempo de Validade,* ou um *Banco de Dados Bitemporal* (quando ambos, tempos de transação e validade são usados). Assim, para permitir uma melhor representação das aplicações do mundo real, o modelo ER temporal utilizado neste artigo considera o último caso, apresentando as propriedades dinâmicos rotuladas por ambos os tempos: transação e validade.

Os modelos de dados ER são, geralmente, construídos utilizando-se de representações gráficas. Portanto, para melhorar o processo de modelagem, as diferenças entre os dois tipos de atributos apresentados são representadas claramente. Por exemplo, utilizando a representação gráfica do ERT, a figura 1 apresenta como os atributos são representados graficamente, identificando as diferenças entre estáticos (a) e dinâmicos (b).

A figura 2 apresenta um diagrama ER Temporal estendido, o qual possui tempo apenas nos atributos. O diagrama foi criado a partir do estudo de caso apresentado.

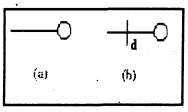

Figura 1 – Símbolos Utilizados no ER Temporal Estendido para Identificar Atributos Estáticos e Dinâmicos

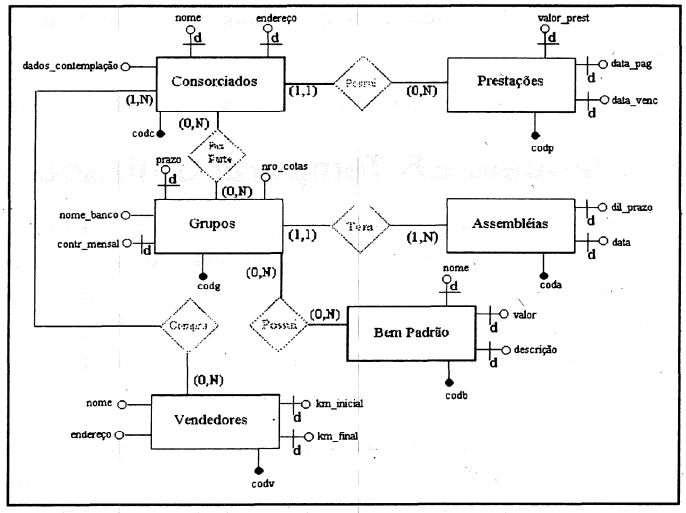

Figura 2 – ERT Representando o Estudo de Caso da Seção 2

4. Mapeamento do Modelo ER Temporal para um Banco de Dados Relacional

Modelos de dados Entidade-Relacionamento (ER) são, geralmente, utilizados nas modelagens conceituais dos projetos de aplicações. E bancos de dados relacionais são empregados para implementar estas aplicações. Como os bancos de dados utilizados não são temporais, a representação dos rótulos temporais implícitos deve ser feita explicitamente, por meio de atributos. Devido a este fato, este artigo analisa como um modelo ER temporal pode ser implementado em um banco de dados relacional.

Cada entidade ou relacionamento apresenta uma tabela base, onde a entidade/relacionamento é identificada e os valores para os *atributos estáticos* são definidos.

Como as propriedades estáticas não mudam com o tempo, não é necessário adicionar informações temporais a elas.

Por outro lado, *propriedades dinâmicas* alteram seu valor com o passar do tempo, e todos os valores assumidos devem ser armazenados no banco de dados. Para que isto seja possível, uma tabela é construída para cada um dos atributos dinâmicos. Onde, para cada um dos valores destes atributos é associado um intervalo de tempo de transação e um de tempo de validade. Esta tabela apresenta, para cada tupla, um valor correspondente ao tipo de dado do atributos, e quatro informações temporais: o início do tempo de validade (*v_timei*), o fim do tempo de validade (*v_timef*), o início do tempo de transação (*t_timei*), e o fim do tempo de transação (*t_timef*).

Quando a evolução de uma aplicação é considerada, a existência de uma entidade/relacionamento também pode ser representada. Isto pode ser feito adicionando rótulos temporais na tabela base, correspondendo ao intervalo de transação e validade da entidade/relacionamento. As quatro informações temporais do últimos parágrafo podem ser utilizadas. Note que somente um intervalo de transação e um intervalo de validade são utilizados para uma entidade/relacionamento, e que eles representam o *lifetime* (tempo de existência) correspondente.

Em relação ao mapeamento dos relacionamentos para o modelo relacional, um procedimento diferente pode ser executado. Uma nova tabela deve ser criada para armazenar os *atributos* dinâmicos de um relacionamento. Esta tabela deve apresentar as entidades envolvidas, o valor dos atributos e as quatro informações temporais (representando os intervalos de transação e validade). A chave primária para esta tabela deve ser composta pelo identificador da entidade e pelo tempo de validade da tupla (duas tuplas de um mesmo atributo não podem ser válidas ao mesmo tempo). Um exemplo de como os atributos estáticos e dinâmicos são armazenados no SGBD relacional é apresentado na figura 3.

Grupos

codg	nro cotas	nome banco	codb
001	100	XXX	010
002	50	YYY	015
003	150	ZZZ	024

Grupos_prazo

codg	prazo	v_timei	v_timef	t_timei	t_timef
002	50	01/03/96	10/09/96	01/03/96	15/08/96
002	60	11/09/96	null	15/08/96	null

Grupos_contr_mensal

codg	contr mensal	v_timei	v_timef	t_timei	t_timef
002	0,02%	01/03/96	10/03/96	01/03/96	15/08/96
002	0,015%	11/09/96	null	15/08/96	null

Figura 3 – Representação de Tabelas da Entidade Grupos em um Banco de Dados Relacional

5. Atualização e Remoção em um Banco de Dados Temporal

Em um banco de dados convencional as operações de atualização e remoção podem ser realizadas sobre qualquer tupla, desde que o usuário possua permissão para tal. Já num banco de dados temporal, o qual deve manter a história das informações, esta facilidade não é comprovada.

Caso um dado seja pesquisado em um banco de dados convencional, todos os usuários interessados por esta informação desejam obter o mesmo resultado. Não deseja-se manter um banco de dados inconsistente. A criação de regras para controlar as atualizações e remoções

para um banco de dados bitemporal visa assegurar a consistência das informações armazenadas, pois recuperam informações tanto do passado, quanto do presente e futuro.

Pode-se analisar quatro regras referentes à operação de *atualização*:

- **atualizar informações somente do futuro**: esta regra deve ser implementada quando deseja-se manter toda a história das informações, sem alteração do passado ou presente. Quando existirem várias tuplas pertencentes ao futuro, deve-se verificar o escopo da regra. Se qualquer tupla do futuro for alterada, deve-se tomar precauções adicionais para manter a consistência. Tais precauções incluem atualizar as respectivas datas de validade e transação das tuplas. Quando somente a última tupla instanciada pode ser atualizada, a tupla anterior a esta deverá receber os respectivos tempo de validade final e tempo de transação final da tupla alterada.
- **atualizar informações do presente e do futuro**: Quando são atualizadas informações válidas no presente deve-se cuidar a granularidade utilizada na implementação do banco de dados temporal. Por exemplo, utiliza-se granularidade de 1 (um) dia. Uma consulta é realizada pela manhã e obtém-se como resultado o valor de R$ 2.000,00 para o atributo "salário" do funcionário "João". Assume-se esta informação como verdadeira. À tarde, um usuário altera esta informação. O funcionário "João" passa a ter o valor de R$ 2.500,00 para o atributo salário. Uma nova pesquisa é realizada obtendo um novo resultado. Nas próximas consultas, o valor válido será o último atualizado. Neste caso, pesquisas em diferentes momentos geraram duas histórias distintas, para um mesmo atributo, em um mesmo dia. Caso a granularidade fosse a hora, a recuperação de dados poderia ser alterada a cada hora, podendo surgir até vinte e quatro resultados diferentes. Quando atualizações do presente forem permitidas, deve-se continuar mantendo a consistência do banco de dados. Os atributos de tempo de validade e tempo de transação das tuplas instanciadas devem ser atualizados. Para atualizar informações do futuro deve-se cuidar, também, a consistência, seguindo as mesmas recomendações dadas para uma atualização do presente.
- **atualizar informações do passado, presente e futuro**: Um banco de dados temporal é utilizado para manter a história das informações. Quando um dado pertencente ao passado pode ser atualizado, sua história não é sempre a mesma. A correção do passado pode fazer com que informações que em algum momento foram válidas, tornem-se inexistentes em consultas futuras. Quando este tipo de correção é possível, deve-se analisar as permissões dos usuário, permitindo, apenas, que pessoas autorizadas alterem tais informações. Deve-se tomar cuidado quanto à consistência. Quando os atributos de tempo de validade e tempo de transação forem atualizados, deve-se cuidar para que tuplas que não eram válidas em um determinado instante, tornem-se válidas quando da atualização. Para as informações do presente e futuro segue-se a descrição da regra anterior.
- **atualizar informações do passado e futuro**: Quando se deseja manter as informações do presente intactas, para que não haja incoerência na recuperação das informações, pode-se propor a alteração de informações apenas do passado ou futuro. Deve-se criar mecanismos e estabelecer restrições para este tipo de opção. Tais restrições devem ser baseadas nas regras anteriores.

Analisando-se estas regras pode-se obter a forma desejada para atualizar as informações. Quanto à *remoção*, o que não deveria ser permitido, mas pode ser um mecanismo importante quando se deseja excluir, do banco de dados, informações muito antigas ou sem relevância. Quando se remove dados do passado, a história é alterada. Para

manter a consistência, deve-se definir restrições para os atributos de tempo de validade e tempo de transação. Quando uma informação do presente é removida, pode-se estar removendo um dado válido, podendo ser utilizado por um outro usuário. Quando se remove informações do futuro deve-se, também, atualizar os tempos de validade e transação. Quando somente a última tupla instanciada, pertencente apenas ao futuro, pode ser removida, a tupla anterior a esta deve tornar-se válida até que outra tupla seja instanciada. Uma solução para manter a história das informações e fazer com que tuplas que iniciaram sua validade no passado e que são válidas no presente sejam teoricamente removidas, é encerrar a sua validade, fazendo com que o atributo "data de validade final" receba a data na qual a tupla deixará de valer.

Para manter a integridade e a consistência das informações, deve-se analisar quais os tipo de restrições, quanto às operações de atualização e remoção, serão utilizadas. Tais restrições devem ser implementadas com a utilização de algum mecanismo do banco de dados utilizado. Um destes mecanismos é o *trigger*, possibilitando que as regras sejam verificadas antes ou depois de qualquer operação.

Através do estudo de caso, apresentado na seção 2, se verificou a necessidade de manter o histórico das informações. Deste modo, tanto a atualização quanto a remoção, para a implementação proposta, utilizam as regras definidas somente para o futuro.

6. Regras para Manutenção de Informações Temporais – Uso de *Triggers*

O mapeamento dos atributos e valores das entidades/relacionamentos não é suficiente para implementar um banco de dados temporal. Existe, também, a necessidade de definir regras para manter a integridade das informações temporais. Estas regras devem ser executadas cada vez que uma nova informação é definida, e quando um valor é alterado ou excluído.

Quando uma operação de inserção (INSERT) é realizada, além do novo valor a ser inserido, o valor temporal correspondente ao tempo de validade inicial da tupla deve, também, ser fornecido, sendo armazenado no atributo referente à validade inicial da tupla (*v_timei*). Um conjunto de regras analisam o valor temporal, para garantir a consistência temporal do banco de dados, porque duas tuplas diferentes, de um mesmo atributo, nunca poderão ser válidas ao mesmo tempo. Assim, estas regras devem ser satisfeitas: (i) o tempo de validade incial (*v_timei*) deve ser maior ou igual à data atual – a validade da tupla não pode iniciar no passado; (ii) validade inicial da tupla (*v_timei*) deve ser maior ou igual ao maior *v_timei* de todas as tuplas inseridas – somente tuplas com *v_timei* maior que o da última tupla inserida serão aceitos; (iii) transação inicial (*t_timei*),transação final (*t_timef*) e validade final (*v_timef*) da tupla, que está sendo inserida, devem ser nulos (valor igual a *null*) – estas três informações temporais são definidas pelo SGBD.

Estas regras foram implementadas através de *triggers* (gatilhos). Para cada uma das operações básicas (INSERT, UPDATE e DELETE), foram implementados dois *triggers*, um acionado antes da execução da operação, denominado BEFORE e outro após a execução, AFTER. Por exemplo, cada vez que uma tupla é inserida no banco de dados, os *triggers before* e *after* são executado. Os caminhos seguidos por uma tupla até ser inserida no banco de dados são apresentados na figura 4.

Caso alguma das regras definidas no *trigger before* não sejam satisfeitas, um erro será gerado, ocasionando o final da execução. Assim, a tupla não será inserida. Caso contrário, realiza a inserção e aciona o *trigger after*.

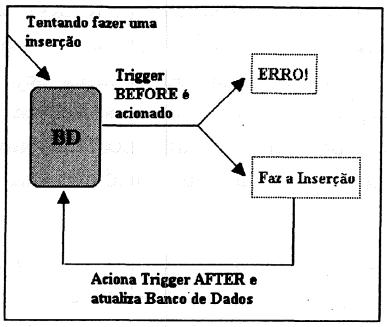

Figura 4 – Inserção de uma Tupla no Banco de Dados

O *trigger after* possui um conjunto de regras que é executado após a inserção de uma tupla no banco de dados, para atualizar as informações temporais das tuplas já inseridas. Primeiro, uma regra encontra qual foi a última tupla inserida, identificada pelo valor *null* do atributo tempo de transação final (*t_timef*), e altera este valor para o tempo de transação atual (data atual). Após, a regra altera o tempo de validade final desta tupla para o tempo de validade inicial da tupla que está sendo inserida. A figura 5 apresenta um esquema simplificado da ação do *trigger* que executa esta regra. Tuplas de uma mesma tabela são utilizadas neste exemplo.

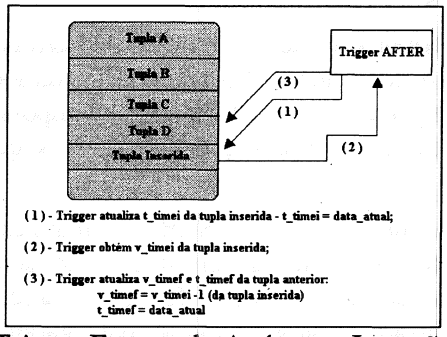

Figura 5 – Ação do *Trigger* Executado Após uma Inserção no Banco de Dados

A operação de atualização (UPDATE) somente é executada após um conjunto de regras serem verificadas. Algumas destas regras são: (i) somente o tempo de validade inicial pode ser atualizado; (ii) um novo tempo de validade inicial anterior ao atual não é aceito; (iii) tempo de validade pertencente ao passado não pode ser modificado; (iv) somente a última tupla definida pode ser atualizada. A atualização do passado pode levar o banco de dados a um estado inconsistente. A figura 6 apresenta o que ocorreria no banco de dados caso qualquer tipo de atualização fosse permitida.

A operação de remoção (DELETE) apresenta regras similares às desenvolvidas para a operação de atualização. Tuplas do passado e tuplas atualmente válidas não podem ser excluídas – como este é um banco de dados temporal, todo o passado é preservado. A operação é, de fato, substituída por um conjunto de regras que substituem a validade da tupla e alteram os rótulos correspondentes: o tempo de validade final da tupla é atualizado para a data atual. Um erro pode ser produzido quando o tempo de validade final de uma tupla for diferente de *null*, significando que esta não pode ser excluída.

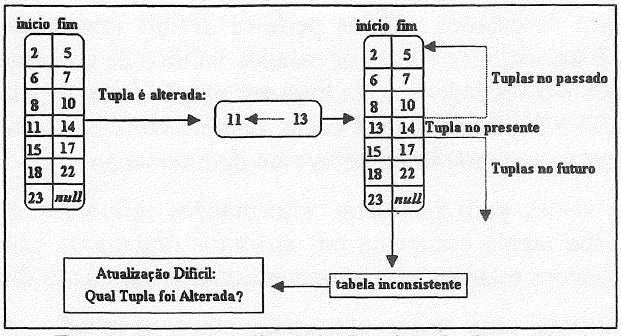

Figura 6 – Atualizando Tuplas do Passado

Após a execução de uma operação de DELETE, um conjunto de ações devem ser executadas no banco de dados para garantir a consistência das informações. O mais importante destas ações é que os rótulos temporais correspondentes aos tempos de validade final e transação final da tupla anterior, devem possuir seu valor igual a *null*, representando que esta é, novamente, a última tupla inserida.

Toda as regras criadas para INSERT, UPDATE e DELETET foram implementadas utilizando-se *triggers*.

7. Regras para Definir Operadores Temporais

Um conjunto de restrições dos atributos temporais pode ser definido para recuperar informações válidas em determinados instantes do tempo. Alguns exemplos de regras para implementação de operadores temporais são apresentados na tabela 1 [CAR 97]:

Tabela 1 – Operadores Temporais

Operador	*Descrição*
Sometime past	Validade inicial de ver menor que a data atual;
Immediately past A	Validade inicial deve ser menor ou igual à data atual e a validade final deve ser maior que a data de ontem ou igual a *null;*
Always past A	Validade inicial de A deve ser igual ao menor tempo de validade inicial definido para o atributo A e a validade final deve ser maior que a data atual;
Sometime future A	A validade final deve ser maior que a data atual;
Immediately future A	Validade inicia deve ser menor ou igual a data-de-amanhã e a validade final deve ser maior que a data-de-amanhã ou igual a *null;*
Always future A	Validade inicial deve ser menor ou igual à data de amanhã e a validade final deve ser igual a *null*;
A since B	Validade inicial de A deve ser menor ou igual à validade inicial de B e a validade final de A deve ser maior ou igual à validade final de B ou, ainda, igual a *null*;
A until B	Validade inicial de A deve ser igual à menor validade inicial definida para a instância e a validade final de A deve ser maior que a validade inicial de B;
A before B	Validade inicial de A deve ser menor que a validade inicial de B;
A after B	Validade inicial de A deve ser maior que a validade inicial de B;

8. Visões e *Stored Functions*

A recuperação de informações, caso não fossem utilizadas visões, geraria consultas muito complexas e muito longas. Quando se deseja recuperar informações ao longo do tempo, deve-se analisar todos os atributos temporais relacionados aos atributos envolvidos na consulta: *v_timei, v_timef, t_timei* e *t_timef.*

A partir da regra de estados válidos pode-se definir uma visão sobre cada uma das principais entidades. Esta visão irá conter os estados válidos da entidade e o período de tempo no qual as informações são verdadeiras. Na implementação feita, foram utilizados, nas visões, os operadores GREATEST e LEAST do Oracle, responsáveis por fornecer o período em que duas tuplas são válidas ao mesmo tempo (intervalo de interseção).

Além de criar visões para recuperar informações referentes a uma entidade, foram criadas visões para cada tabela composta por atributos dinâmicos, pois muitas vezes não se deseja todas as informações relacionadas a uma entidade e sim, a um único atributo.

Uma função retorna uma única expressão como resultado. Assim, para as *stored functions*, estipulou-se utilizar a função de agregação COUNT, retornando um número inteiro (zero – não satisfaz; 1 satisfaz) ou, na utilização dos operadores *always past* e *always future*, a data de validade inicial da tupla.

Foram criadas *stored functions* para todos os operadores temporais da seção 7.

9. Recuperação de informações temporais

Várias são as consultas possíveis num banco de dados temporal, pois recuperam o momento atual, o passado e o futuro. Na implementação realizada, a criação de operadores especiais possibilitou uma recuperação rápida de informações puramente temporais, importantes na aplicação modelada.

Todos os operadores da seção 7 foram anexados à linguagem de consulta do Oracle (PL/SQL). Assim, o PL/SQL foi estendido para suportar diferentes tipos de consultas e recuperar um número maior de informações. Por exemplo, a execução do operador ***Sometime Future***, o qual recupera a tupla que é válida em algum momento do futuro, é apresentada na figura 7.

A primeira parte da figura apresenta a execução da função, obtendo-se como resultado o valor 1. Portanto, a consulta foi satisfeita, ou seja, o atributo "GolI" é válido em algum momento do futuro. A Segunda parte da figura apresenta a consulta sobre a tabela *Bem_nome*, verificando que a data de validade final, da tupla identificada por "GolI", não foi determinada, ou seja, é igual a *null*.

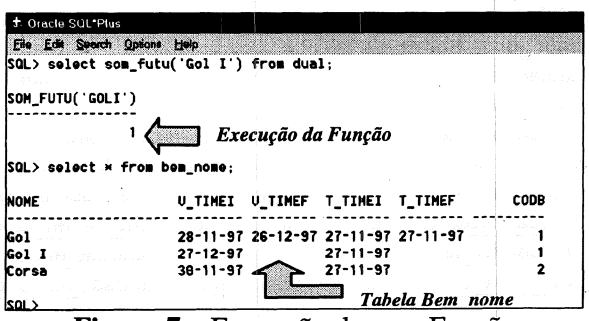

Figura 7 – Execução de uma Função

10. Conclusão

Através da análise do estudo de caso concluiu-se que a utilização de um banco de dados temporal seria apropriado para suprir as necessidades da empresa. Esta análise foi realizada

através do estudo do *software* já existente, o que levou à constatação da existência de muitos dados duplicados, pois armazenavam a história de muitas informações. A modelagem do estudo de caso para um diagrama ER com as informações temporais, possibilitou a visualização de todas as entidades, bem como seus atributos dinâmicos e estáticos.

Para realizar a implementação, regras foram criadas para a manutenção das informações temporais. Tais regras descrevem as restrições que devem ser aplicadas sobre as operações realizadas, no banco de dados, para que a história seja preservada. Através delas constatou-se a necessidade de utilização de algum mecanismo, que fosse invisível ao usuário e que mantivesse o banco de dados estável e consistente. O mecanismo encontrado para suprir tal necessidade foi o *trigger*. Através dele, as regras foram implementadas e automaticamente executadas.

Além do *trigger*, outros mecanismos foram utilizados. Um deles foi a *stored function* a qual possibilitou a implementação de operadores temporais bastante úteis no processo de recuperação de informações. Após longa análise sobre a utilização de *stored procedures* chegou-se à conclusão que poderiam ser utilizadas para facilitar a descrição das tabelas nos casos de uma nova inserção de tupla ou de uma atualização.

Outro mecanismo muito utilizado foram as visões. Caso não fossem construídas, consultas muito extensas e difíceis teriam que ser criadas pelo usuário, pois o mapeamento gera um número bastante grande de tabelas e as regras temporais são muito complexas. Além disso, as visões possibilitam a junção da tabela dos atributos estáticos com a tabela dos atributos dinâmicos reconstruindo, assim, a entidade original.

Um problema encontrado na definição das regras para manutenção das informações refere-se às operações de remoção e atualização. Isto deve-se ao fato do mapeamento estar utilizando intervalos de tempo contínuos para representação da história das informações. A operação pode quebrar esta seqüência e os tempos dos atributos teriam que ser alterados para manter o intervalo. A atualização do banco de dados, nestes casos, torna-se quase impossível, pois não se sabe qual tupla foi alterada e/ou removida, impossibilitando a localização das tuplas que devem ser atualizadas.

A implementação de um modelo temporal em um SGBD convencional é totalmente viável, pois existe um conjunto de regras possíveis que podem ser utilizadas para restringir as operações a serem realizadas sobre o mesmo. Além disso, os diferentes estados dos objetos podem ser recuperados de forma simples e com o mesmo poder de expressão dos modelos de dados temporais.

Referências Bibliográficas

[CAR 97] CARVALHO, Tanisi Pereira de. *Implementação de Consultas para um Modelo de Dados Temporal Orientado a Objetos*. Porto Alegre: Departamento de Ciência da Computação - UFRGS, 1997. Dissertação de Mestrado.

[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, Outubro, 1995.

[CLI 87] CLIFFORD, J.; CROCKER, A. The historical relational data model (HRDM) and algebra based on lifespans. *IEEE Transactions on Software Engineering*, New York, v.14, n.7, July 1988. Trabalho apresentado na International Conference on Data Engineering, 1987, Los Angeles, California.

[CLI 95] CLIFFORD, J.; TUZHILIN, A. (Eds.) *Recent Trends in Temporal Databases*. Great Britain: Springer, 1995. 360p.

[EDE 93] EDELWEISS, N.; OLIVEIRA, J. Palazzo M.; PERNICI, B. *An Object-Oriented Temporal Model.* Proceedings of the 5th International Conference on Advanced Information System Engineering - CAISE'93, June 8-11, 1993, Paris. Heidelberg: Springer-Verlag, 1993. p.397-415. (Lecture Notes in Computer Science 685).

[EDE 94] EDELWEISS, N. Modelagem de Aspectos Temporais de Sistemas de Informação. In: ESCOLA DE COMPUTAÇÃO, 9.,1994, Recife. *Anais.* Recife: UFPE, 1994.

[EDE 98] EDELWEISS, N. *Bancos de Dados Temporais: Teoria e Prática.* XVII Jornada de Atualização em Informática - JAI'98 - XVIII Congresso da SBC. Belo Horizonte: agosto 1998.

[ELM 93] ELMASRI, R.; WUU, G. T. J.; KOURAMAJIAN, V. *A temporal model and query language for EER Databases.* In: TANSEL, A. et al. (Eds.). Temporal databases: theory, design and implementation. Redwood City: The Benjamin/Cummings Publishing, 1993. p. 212-229.

[GAD 88] GADIA, S. *A homogeneous relational model and query language for temporal databases.* ACM Transactions on Database Systems, New York, v.13, n.4, p.418-448, Dec.1988.

[HUB 97] HUBLER, Patrícia Nogueira. *Armazenamento e Recuperação de Informações Temporais em SGBDs Convencionais.* Gravataí/RS: Universidade Luterana do Brasil, 1997. Trabalho de Conclusão.

[JEN 97] JENSEN, C.S. *Tutorial on Temporal Databases.* SBBD'97, Fortaleza, CE, 1997.

[KAF 92] KAFER, W.; SCHONING, H. *Realizing a temporal complex-object data model.* Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2-5, 1992, San Diego. p.266-275.

[LOR 88] LORENTZOS, N.A.; JOHNSON, R.G. *Extending relational algebra to manipulate temporal data. Information Systems.* v.13, n.3, p.289-296, 1988.

[LOU 91] LOUCOPOULOS, P.; THEODOULIDIS, C.; WANGLER, B. *The entity relationship time model and conceptual rule language.* Proceedings of the 10th International Conference on the Entity Relationship Approach, 1991, San Mateo, California.

[NAV 89] NAVATHE, B.; AHMED, R. *A Temporal relational model and a query language.* Information Sciences, v.49, p. 147-175, 1889.

[OZO 95] ÖZSOYOGLU, G.; SNODGRASS, R. T. Temporal and real-time databases: a survey. *IEEE Transactions on Knowledge and Data Engineering*, New York, v.7, n.4, p.513-532, Aug. 1995.

[OZS 95] ÖZSU, M.T. et al. TIGUCAT: A Uniform Behavioral Objectbase Management System. *The VLDB Journal*, v.4, p.100-147, Aug. 1995.

[ROS 91] ROSE, E.; SEGEV, A. *TOODM: A Temporal object-oriented data model with temporal contraints.* Proceedings of the 10th International Conference on the Entity Relationship Approach, Oct. 1991, San Mateo, California.

[SAR 90] SARDA, N. L. Extensions to SQL for historical databases. *IEEE Transaction on Knowledge and Data Engineering*, v.2, n.2, June 1990.

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

[SU 91] SU, S.Y.W.; CHEN, H.-H. M. *A Temporal knowledge representation model OSAM*/T and its query language OQL/T.* Proceedings of the 17th International Conference on Very Large Data Bases, Sept. 1991, Barcelona. p.431-442.

[TAN 86] TANSEL, A.U. Adding time dimension to relational model and extending relational algebra. Information Systems, v.11, n.4, p.343-355, 1986.

[TAN 93] TANSEL, A.U. et al. (eds.) *Temporal Databases: Theory, Design, and Implementation.* Redwood City, California: Benjamin/Cummings, 1993.

[TAU 91] TAUZOVICH, B. *Towards temporal extensions to the entity-relationship model.* Proceedings of the 10th International Conference on the Entity Relationship Approach, 1991, San Mateo, California.

[WUU 93] WUU, G. T. J.; DAYAL, U. *A uniform model for temporal and versioned object-oriented databases.* In: TANSEL, A. et al. (Eds.). Temporal databases: theory, design and implementation. Redwood City: The Benjamin/Cummings Publishing, 1993. p.230-247.

[ZAN 97] ZANIOLO, C.; CERI, S.; FALOUTSOS, C.; SNODGRASS, R.T.; SUBRAHMANIAN, V.S.; ZICARI, R. *Advanced Database Systems.* San Francisco, CA: Morgan Kaufmann Publishers, 1997. 574p.