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

TIAGO SPERB CASSOL

Um estudo sobre alternativas de representação de dados temporais em bancos de dados relacionais

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser

Porto Alegre, dezembro de 2012.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cassol, Tiago Sperb

Um estudo sobre alternativas de representação de dados temporais em bancos de dados [manuscrito] / Tiago Sperb Cassol. – 2012.

105 p.:il.

Orientador: Carlos Alberto Heuser;

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2012.

1.Informática. 2.Bancos de Dados Temporais. I. Heuser, Carlos Alberto. II. Título.

AGRADECIMENTOS

Algumas pessoas foram essenciais para a conclusão desse trabalho. E não poderia deixar de lembra-las nesse espaço.

Em primeiro lugar, meus pais, Maria Aparecida e Celito, verdadeiros pilares de sustentação para mim. Eles me ensinaram valores que hoje pontuam a minha vida, como amor ao próximo, humildade, dedicação e correção. Se durante minha vida conseguir ser metade da pessoa que eles são, posso considerá-la um sucesso. Como não poderia deixar de ser, eles foram essenciais nos momentos em que apenas a minha força e determinação pareceu não ser suficiente. Dedico a eles a conclusão desse trabalho.

À minha irmã Clarissa, com quem pude contar sempre que precisei. Na equação de muitas variáveis que descreve a minha vida, ela é uma constante, que sempre está ali somando nos momentos cruciais.

Ao meu orientador Carlos Alberto Heuser, que durante essa longa jornada do mestrado foi a definição perfeita da palavra “orientador”, me guiando enquanto eu navegava pelas águas tortuosas e intempestivas da pesquisa, e sem o qual eu não teria conseguido concluir essa jornada tão árdua.

A todos os meus amigos, que compreenderam os inúmeros momentos em que estive ausente porque precisava concluir um experimento ou terminar de escrever um artigo ou um capítulo da dissertação.

Tenho a sorte de ter amigos demais pra citar todos aqui, mas não poderia deixar de mencionar especialmente os que acompanharam essa caminhada mais de perto: Lucas Vianna, Paulo Mello, Rodrigo Geiger, Leonardo Decker, Daniel Carvalho, Cristiano Basso e Flávio Knob. Eles estiveram bem próximos enquanto eu carregava o peso das obrigações, nada mais justo que dividir com eles os louros da vitória.

Por fim, uma menção especial também aos amigos do “Taquaraço”, especialmente aos amigos do “Taquaraço-B” dos jogos de futebol nas terças, que muitas vezes providenciaram o escape necessário em momentos de estresse intenso. Os longos anos de convivência, com alguns desde o início da faculdade, com outros a partir dos campeonatos que vieram depois da formatura, parecem estreitar cada vez mais os nossos laços de amizade.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema de gerência de banco de dados
UFRGS	Universidade Federal do Rio Grande do Sul
CPF	Cadastro de Pessoas Físicas
RG	Registro Geral
XML	e*Xtensible Markup Language*
ER	Entidade-Relacionamento
SQL	Structured Query Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Informações temporais estão presentes numa ampla gama de aplicações. Praticamente qualquer aplicação possui pelo menos um campo que contém dados temporais como datas ou *timestamps*. Entretanto, bancos de dados tradicionais não tem um suporte amplo para armazenamento e consulta sobre esse tipo de dados eficientemente, e SGBDs com suporte nativo para dados temporais raramente estão disponíveis para os desenvolvedores de sistemas.

Na maior parte do tempo, bases de dados comuns são usadas para armazenar dados das aplicações, e quando dados temporais são necessários, eles são gerenciados utilizando o pobre suporte oferecido por SGBDs relacionais tradicionais. Dito isso, o projetista da base de dados precisa confiar em um bom *design* de esquema para que a dificuldade natural enfrentada ao lidar com dados temporais possa ser minimizada.

Enquanto algumas escolhas de *design* podem parecer óbvias, outras são difíceis de avaliar apenas com uma análise superficial, necessitando experimentação antes de serem aplicadas ou não. Por exemplo, em vários casos pode ser difícil de medir o quanto uma determinada escolha de *design* vai afetar o consumo de espaço em disco, e quanto essa mesma escolha afetará a performance geral. Esse tipo de informação é necessária para que o projetista da base de dados seja capaz de determinar se , por exemplo, o aumento no consumo de espaço em disco gerado por uma escolha específica é aceitável por conta da melhora de performance que ela oferece.

O problema é que não há estudo que analise as escolhas de *design* disponíveis, fazendo uma análise através de dados concretos. Mesmo quando é fácil identificar, dentre duas escolhas, qual tem performance melhor em um determinado critério, é difícil mensurar o *quão* melhor a escolha melhor se sai, e se algum efeito colateral trazido por ela é aceitável. Ter dados concretos para suportar esse tipo de decisão permite ao projetista da base de dados fazer escolhas que se enquadram melhor no contexto da sua aplicação.

O objetivo desse trabalho é analisar algumas escolhas de design comuns para representar e gerenciar dados temporais em SGBDs relacionais tradicionais, provendo direcionamento sobre qual alternativa se enquadra melhor em cada situação onde dados temporais são necessários. Dados concretos sobre cada uma das alternativas estudadas são gerados e analisados e conclusões são obtidas a partir deles.

Palavras-Chave: bancos de dados relacionais, bancos de dados temporais, tempo de validade, tempo de transação, registro de alterações, entidades temporais.

A study on alternatives to represent temporal data on relational databases

ABSTRACT

Temporal information is present on a wide range of applications. Almost every application has at least one field that contains temporal data like dates or timestamps. However, traditional databases don't have a comprehensive support to storage and query this kind of data efficiently, and DBMS with native support for temporal data are rarely available to system developers.

Most of the time, regular databases are used to store application data and when temporal data is needed, it is handled using the poor support offered by standard relational DBMS. That said, the database designer must rely on good schema design so that the natural difficulty faced when dealing with temporal data on standard relational DBMS can be minimized.

While some design choices may seem obvious, others are difficult to evaluate just by looking at them, therefore needing experimentation prior to being applied or not. For example, in several cases it might be difficult to measure how much will a specific design choice affect the disk space consumption, and how much will this same design choice affect overall performance. This kind of information is needed so that the database designer will be able to determine if, for example, the increased disk space consumption generated by a given choice is acceptable because of the performance enhancement it gives.

The problem is that there is no study that analyses the design choices available, analyzing them through concrete data. Even when it is easy to see which of two design choices perform better in a given criterion, it is hard to see *how* better the better choice does, and if any other side-effect it has is acceptable. Having concrete data to support this kind of decision allows the database designer to make the choices that suits his application's context best.

The objective of this work is to analyze several common design choices to represent and handle different kinds of temporal data on standard SQL DBMS, providing guidance on which alternative suits best each situation where temporal data is required. Concrete data about each of the studied alternatives are generated and analyzed, and conclusions are drawn from them.

Keywords: relational databases, temporal databases, valid-time, transaction-time, tracking logs, temporal entities.

1 INTRODUÇÃO

Lidar com dados temporais em bases de dados relacionais é um problema recorrente. Muitas aplicações são temporais por natureza, o que torna o uso e representação de dados temporais em bases de dados muito comum. Tanto aplicações simples como, por exemplo, um sistema de gerência de uma vídeo-locadora, que armazene empréstimos de DVDs e suas respectivas durações, como aplicações mais complexas, tal qual sistemas de geoposicionamento, precisam armazenar informações sobre tempo e na maioria dos casos isso é feito em bases de dados relacionais.

Existe muita pesquisa na área de bases de dados temporais. Essa área de pesquisa é vibrante desde o inicio dos anos 90 (SNODGRASS, 1990) e até hoje muitos trabalhos na área são desenvolvidos (MKAOUAR, 2011; AHMED, 2012; ALI 2012). Entretanto, mesmo com a quantidade de trabalhos gerados, há pouco foco na situação que normalmente se apresenta para os projetistas e desenvolvedores de aplicações de bases de dados: lidar com dados temporais em bancos de dados puramente relacionais, nos SGBDs mais tradicionais e mais usados nos projetos de sistemas de TI. A maioria dos trabalhos foca ou em extensões para esses sistemas, colocando uma camada intermediária entre a aplicação e a base de dados puramente relacional, ou em extensões e alterações sobre sistemas de bancos de dados de código aberto para facilitar a manipulação de dados temporais nessas bases relacionais.

Como os SGBDs tradicionais oferecem pouco suporte para operações e armazenamento de dados temporais por padrão, ao lidar com dados temporais nesses SGBDs o projetista ou desenvolvedor de aplicações de bases de dados normalmente tem duas opções. A primeira delas é usar esse suporte limitado oferecido pelos SGBDs relacionais tradicionais e depender de um esquema bem construído para minimizar eventuais dificuldades, ou utilizar uma das possibilidades mencionadas anteriormente como alternativas para um suporte melhor a dados temporais em bases de dados relacionais, como tradutores de consultas (expressas em uma linguagem que simplifique as mesmas quando envolvem tempo) ou extensões de terceiros.

É neste contexto, da necessidade do uso de dados temporais sobre bases de dados puramente relacionais, que se enquadra o estudo desenvolvido neste trabalho. Esse capítulo dá uma visão geral do trabalho e está estruturado da seguinte forma:

- Na seção 1.1 é descrito o problema a ser abordado pelo trabalho;
- Na seção 1.2 são apresentadas as motivações do trabalho;
- Na seção 1.3 são apresentados os objetivos gerais do trabalho;
- Por fim, na seção 1.4 é descrita a organização do restante do trabalho

1.1 O Problema

Quando o contexto de uma determinada aplicação de base de dados exige que o projetista da base de dados lide com dados temporais, ele tem um grande problema nas mãos e algumas alternativas para resolvê-lo.

O problema parte do fato de que bases de dados puramente relacionais não são exatamente preparadas para lidar com dados temporais. Seu suporte para esse tipo de dado é limitado na especificação SQL-92, e, mesmo em especificações mais recentes, como SQL:2011, ainda é insuficiente. Soma-se a isso, ainda, o fato de que os SGBD tradicionais são seletivos em relação a operações e tipos de dados temporais que implementam.

Com isso, o projetista deve tentar otimizar o seu esquema para lidar com esses dados da melhor maneira possível, de acordo com o contexto em que sua aplicação está inserida. Por isso, ao definir a base de dados, algumas questões costumam surgir:

- Quais campos de quais tabelas são temporais?
- É possível separar os campos de uma entidade em campos temporais e atemporais? Vale a pena fazer isso e introduzir uma junção entre tabelas nas consultas? É possível economizar espaço em disco relevante fazendo isso? E o tempo de processamento adicional compensa?
- Existe alguma entidade para a qual seja necessário, para todos os seus campos, armazenar seus valores enquanto eles variam no tempo?
- Como representar a validade de uma informação? Deve-se armazenar apenas o ponto em que uma informação passou a ser válida ou armazenar o inicio e o fim da validade de cada registro? Qual alternativa simplifica mais a escrita de consultas?
- Como garantir as restrições de integridade das tabelas ao longo do tempo? Será que os procedimentos necessários para isso irão ter impacto relevante na performance do banco?
- Ao fazer um registro de tudo que é alterado no banco, devem-se utilizar imagens anteriores (estado do registro antes da alteração) ou posteriores (estado do registro após a alteração)? Existe alguma diferença de performance entre elas?

Esses são apenas exemplos de questionamentos que surgem para o projetista da base de dados quando ele decide como modelar os dados temporais que sua aplicação requer. Alguns desses questionamentos são simples de resolver e sua solução é intuitiva. Outros, entretanto, são complexos de resolver sem uma experimentação prévia.

1.2 Motivações

Enquanto a maioria dos trabalhos presentes na literatura foca em soluções que pressupõe a existência de “anexos” às bases relacionais tradicionais, com camadas intermediárias entre o SGBD tradicional e a aplicação, essa é uma possibilidade que muitas vezes não se encontra disponível para o projetista ou desenvolvedor dessas aplicações.

Dessa forma, se faz necessário um estudo minucioso sobre as alternativas de modelagem disponíveis para o projetista em uma situação em que ele dispõe apenas da base de dados puramente relacional para implementar a sua aplicação. Assim, qualquer decisão tomada por ele será suportada por dados concretos gerados a partir de experimentação, e não por meio apenas de suposições ou experiências prévias.

1.3 Objetivo

Levando-se em consideração as motivações apresentadas na seção anterior, o objetivo desse trabalho é analisar algumas das diversas alternativas que o desenvolvedor de aplicações de bases de dados tem ao lidar com dados e consultas temporais sobre suas aplicações. A intenção é que os resultados obtidos possam prover o suporte experimental para as decisões do projetista mencionadas na seção anterior, permitindo que ele saiba, com base em dados concretos, qual a melhor alternativa a seguir em cada caso.

Esse objetivo é alcançado através da experimentação de algumas das alternativas de implementação de dados temporais em bases de dados relacionais existentes na literatura, que são detalhadas ao longo desse trabalho. Nesses experimentos, é realizada a implementação de uma base de dados para servir as necessidades de um contexto prático onde se faz necessário o uso de dados temporais. Essa implementação é feita para todas as alternativas que são detalhadas no trabalho. As alternativas são analisadas utilizando critérios objetivos, com dados concretos, e subjetivos, como análise da simplicidade de escrita das consultas e procedimentos relativos a cada alternativa. Através da análise de cada alternativa de acordo com os critérios estabelecidos é possível oferecer ao desenvolvedor uma ideia clara das vantagens e desvantagens de cada alternativa em cada situação.

1.4 Organização do Texto

Esse trabalho é organizado da seguinte maneira:

- O capítulo 2 apresenta conceitos introdutórios de dados temporais, e como esses se aplicam e são ou não suportados nas diferentes especificações da linguagem SQL e SGBD tradicionais;
- O capítulo 3 apresenta as alternativas estudadas para representação de dados temporais em bases de dados relacionais, e situações onde cada uma delas se encaixa;
- No capítulo 4, essas mesmas alternativas são estudadas através dos experimentos mencionados anteriormente. Esses experimentos são explicados e detalhados ao longo do capítulo, assim como as conclusões derivadas deles;
- Por fim, o capítulo 5 apresenta as conclusões gerais do trabalho e possíveis objetivos a serem perseguidos através de trabalhos futuros.

2 CONCEITOS TEMPORAIS

Para enfrentar os problemas decorrentes da necessidade de se armazenar, alterar e recuperar informações temporais em uma base de dados relacionais é necessário familiaridade com os conceitos referentes à dimensão tempo em uma base de dados e também com o suporte oferecido pelas bases de dados à manipulação de dados temporais.

Nesse capítulo serão cobertos os conceitos básicos referentes a dados temporais em bases de dados relacionais e o que é suportado ou não na linguagem SQL, de acordo com a sua especificação.

2.1 Tipos de dados temporais

Dados temporais fundamentalmente podem ser de três tipos: instantes, intervalos ou períodos (SNODGRASS, 1999-a).

Instantes são pontos específicos no tempo. Instantes podem estar no presente (instante corrente), no futuro (instantes cronologicamente posteriores ao instante corrente) e no passado (instantes cronologicamente anteriores ao instante corrente) (SNODGRASS, 1999-a). Instantes podem ser associados a outras informações, como o momento a partir do qual algo passou a ser válido ou o momento em que algo deixou de ser válido em um determinado contexto. Um exemplo de instante é: “28 de setembro de 2012”. Nesse exemplo, a granularidade cronológica é *dia*. Caso o instante fosse “28 de setembro de 2012, às 20 horas e 23 minutos”, a granularidade cronológica seria o segundo.

Intervalos são porções de tempo não específicas. Intervalos, ao contrário de instantes, são relativos. Além disso, intervalos também tem direção cronológica, ou seja, podem ser referentes ao passado ou ao futuro (SNODGRASS, 1999-a). Um exemplo de intervalo seria “daqui a três meses” (ou seja, três meses no futuro, em relação ao instante corrente). Outro exemplo de intervalo poderia ser “trinta dias antes do natal”, ou seja, trinta dias no passado em relação ao dia 25 de Dezembro. A distância entre dois instantes também é um intervalo.

Por fim, períodos são porções de tempo que tem início e fim bem definidos, iniciando em um instante e terminando em outro no futuro (SNODGRASS, 1999-a). Um exemplo de período seria: “Início em 28 de novembro de 2012, término em 31 de novembro de 2012”.

Esses tipos de tempo, períodos, intervalos e instantes, geralmente são associados aos dados das entidades para representar a validade dos mesmos.

2.2 Tipos de tempo

Como visto nas sessões anteriores, informações contidas em bases de dados relacionais podem conter dados temporais associadas a elas, indicando algo como sua validade ou o momento em que foram inseridas na base de dados. Essa informação de tempo em geral é classificada em duas categorias: tempo de validade e tempo de transação.

O tempo de validade representa o período em que uma informação é válida no mundo real. Ou seja, uma tabela em que haja dados temporais associados a um atributo, informando que o valor daquele atributo no registro é valido de um instante X até outro instante futuro Y, possui informações de tempo de validade daquele atributo. Na base de dados, esse período usualmente é representado utilizando um instante (válido de X até o início da validade do próximo registro referente ao mesmo atributo ou entidade, em ordem cronológica) ou dois instantes (válido de X até Y).

Já o tempo de transação diz respeito ao instante em que dados foram inseridos na base de dados. Por exemplo, o registro com identificador igual a Z teve o valor do campo W alterado para N no instante U.

Em geral, apenas o tempo de validade ou o tempo de transação é armazenado, mas existem casos onde ambos são utilizados. Algumas vezes também se assume o tempo de transação e o tempo de validade como sendo iguais, ou seja, a partir do momento que um dado é inserido na base de dados, ele passa a ser válido na vida real também (prática comum em sistemas que lidam diretamente com preços e produtos, ao invés de simplesmente serem uma representação da realidade). Entretanto, em geral apenas o tempo de validade é utilizado.

Um exemplo prático de um texto que descreve um registro com informações temporais é o seguinte: “A inscrição do aluno com CPF 001.001.001-11 no curso de Programação vale do dia 2 de Janeiro de 2012 até o dia 15 de outubro do mesmo ano. Essa informação foi inserida na base de dados às vinte e uma horas e três minutos do dia 13 de dezembro de 2011”. Nesse texto, o tempo de validade do registro referente à inscrição do aluno é o período que compreende do dia 2 de janeiro de 2012 até o dia 15 de outubro do mesmo ano. O tempo de transação é o instante referente ao dia 13 de dezembro de 2011.

2.3 Suporte temporal na linguagem SQL

Apesar de simples, os conceitos temporais apresentados nas sessões anteriores não são totalmente suportados pela linguagem SQL. Além disso, mesmo os que são suportados não são implementados em muitos SGBDs comerciais, especialmente os conceitos mais recentes incorporados na linguagem.

O suporte inicial para dados temporais em bases de dados relacionais foi incorporado na linguagem SQL no padrão SQL-92. Nesse padrão, são suportados intervalos e instantes e algumas operações sobre eles. Já períodos precisam ser simulados utilizando uma combinação de instantes. Instantes são suportados como tipos de coluna através de alguns diferentes tipos de dados, cada um deles representando uma granularidade diferente. Os tipos são: DATE, TIME, TIMESTAMP, TIME WITH TIMEZONE e TIMESTAMP WITH TIMEZONE. Além disso, foi adicionado suporte para algumas operações temporais, como adição e subtração de dados temporais e comparações entre períodos compostos por dois instantes através do operador

OVERLAPS. É possível, por exemplo, somar instantes com intervalos, intervalos com intervalos, datas (tipo DATE, sem informação de hora, minuto e segundo) com horas do dia (tipo TIME, cujos valores compreendem os horários possíveis dentro de um dia), entre outras. Também existe suporte para extrair valores específicos de instantes, através de palavras reservadas como YEAR, MONTH e SECOND, por exemplo. É possível ainda construir dados temporais através de *strings*, como por exemplo no comando "TIMESTAMP '2012-01-01'", que irá retornar o instante referente ao dia primeiro de janeiro de 2012, com os dados de hora, minuto e segundo zerados, e será do tipo TIMESTAMP. Por fim, existem ainda palavras reservadas para instantes específicos, como CURRENT_DATE (dia atual), CURRENT_TIME (hora do dia atual) e CURRENT_TIMESTAMP (dia e hora atuais). Uma descrição completa da linguagem SQL-92 e seu suporte temporal pode ser encontrada em (MELTON, 1993).

Apesar dos vários tipos de dados e operações, o suporte da linguagem SQL-92 para os conceitos temporais ainda era bastante precário, uma vez que conceitos básicos como períodos e o conceito de "agora" para colunas não era implementado (necessário quando não se sabe o instante final do tempo de validade de um dado). E mesmo assim muitas vezes apenas parte dos conceitos suportados é implementada em SGBDs comerciais (o operador OVERLAPS, por exemplo, fica de fora de alguns deles). Conceitos como tempo de validade e tempo de transação, por exemplo, também não são suportados.

Recentemente, em Dezembro de 2011 (KULKARNI, 2012), foi adotado formalmente o padrão SQL:2011, e ele contém algumas adições importantes em relação a dados temporais. Foi adicionado suporte para tabelas com tempo de validade e de transação embutidos (um dos dois ou ambos), suporte para chaves primárias e estrangeiras temporais (cuja importância será estuada no capítulo 3 desse trabalho) além de um suporte maior para consultas temporais. Entretanto, mesmo com o suporte do padrão SQL:2011, poucos SGBDs implementam algumas dessas novidades, obrigando o projetista a utilizar uma extensão ou camada intermediária entre a base de dados tradicional e a aplicação ou a utilizar o suporte disponível no SGBD para implementar dados temporais.

Alguns SGBDs comerciais oferecem soluções próprias que comtemplam alguns dos conceitos mencionados nesse trabalho. Os sistemas Teradata 13.10, IBM DB2 10, Oracle 11g *workspace manager* e PolarLake DM oferecem suporte para tempo de transação e tempo de validade, por meio de extensões ou nativamente no SGBD.

Entretanto , usualmente o projetista apenas tem à sua disposição uma base de dados relacional tradicional. É com o suporte existente nela, que na grande maioria dos casos comporta apenas parte das definições especificadas na linguagem SQL-92, que o projetista deve desenhar e implementar a sua base de dados. No capítulo à seguir, são detalhadas as alternativas encontradas na literatura escolhidas para cobrir nesse trabalho.

3 ALTERNATIVAS DE REPRESENTAÇÃO DE DADOS TEMPORAIS

Dados temporais em bases de dados podem ser definidos em nível de atributos (*atribute-timestamping*) ou entidades (*tuple-timestamping*) presentes nas relações (NOH, 2008-a; BERGAMASCHI, 1998; ATAY, 2010). Nesse trabalho, são utilizadas as seguintes definições: quando um atributo de uma entidade possui valores que variam no tempo, esse atributo é chamado um atributo temporal (TANSEL, 2008). Já quando, no projeto da base de dados, é definida uma temporalidade para o estado da entidade como um todo, essa entidade é chamada de entidade temporal (PARENT, 1999). Essas opções de modelagem aparecem com frequência na literatura, e por isso foram escolhidas para o estudo realizado nesse trabalho.

Para representar dados temporais, sejam eles em nível de atributo ou entidade, deve-se definir um modelo para representar os dados. O modelo escolhido impactará a base de dados gerada em vários aspectos. Alguns modelos favorecem espaço em disco em detrimento de desempenho em alguns tipos de consultas, enquanto outros oferecem melhor desempenho para atualizações, mas desempenho pior em recuperação de dados. Além disso, é preciso considerar também a simplicidade existente ao criar consultas para os mais diversos fins em uma base de dados, uma vez que é natural que aplicações de bases de dados sofram atualizações e alterações de manutenção de acordo com as necessidades atendidas pela aplicação. Entretanto, sem uma análise profunda sobre as alternativas possíveis é complicado saber qual a melhor escolha em termos de projeto de uma base de dados. Normalmente, é possível instintivamente saber qual aspecto cada decisão de projeto irá afetar, mas dificilmente é possível quantificar esses impactos para fazer uma comparação objetiva entre eles.

O objetivo desse capítulo é apresentar conceitualmente as alternativas estudadas para modelar dados temporais, tanto em nível de atributos quanto em nível de entidades. Isso se faz necessário para introduzir o estudo realizado no capítulo seguinte, onde são apresentados experimentos sobre essas alternativas, para quantificar e comparar o impacto de cada decisão de modelagem sobre os vários aspectos de uma base de dados.

3.1 Atributos temporais

Quando o valor de um atributo da relação varia no tempo, seja ele de transação ou de validade, esse atributo é determinado um atributo temporal, ou seja, cada valor desse atributo ao longo do tempo tem um tempo de validade associado. Algumas vezes é possível modelar apenas alguns dos atributos de uma entidade como temporais, sendo essa uma decisão de projeto, com base nas possíveis necessidades de uma aplicação de

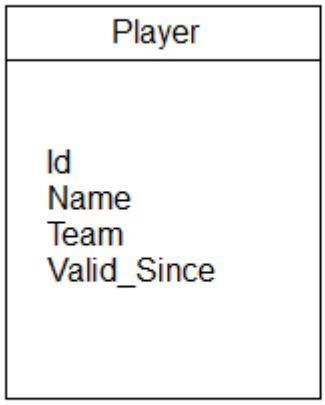
base de dados. Nesse caso, atributos temporais e atemporais coexistem na mesma entidade.

Quando um atributo é determinado temporal, existem algumas alternativas para representar e armazenar os estados deste atributo ao longo de sua variação no tempo. As alternativas comumente apresentadas na literatura envolvem a escolha entre utilização de instantes ou de períodos para a representação do atributo temporal (NOH, 2008-b;O'CONNOR, 2002). Em geral não se usam intervalos para essa finalidade pois usualmente os dados armazenados na base precisam que o seu tempo de validade seja associado a pontos específicos no tempo, algo que não existe em intervalos.

3.1.1 Modelo baseado em instantes

Ao se utilizar instantes pra representar o tempo de validade da informação de um atributo, apenas um campo de tempo é associado ao atributo. Assim, esse campo de tempo representa o momento em que o valor de um atributo passa a ser válido como um fato do mundo real. Esse valor é válido até que o atributo mude de valor novamente ao longo do tempo. Nesse caso, é adicionado um novo registro na base de dados com o valor do atributo atualizado.

Por exemplo, no caso de uma base de dados que modele as informações de um jogador de futebol, ela pode conter um atributo que representa o time do jogador. Como essa é uma informação que se altera ao longo do tempo, a aplicação de base de dados pode ter interesse em armazenar os valores desse dado ao longo do tempo. Nesse caso, a base de dados poderia ser modelada de acordo com o modelo ER na figura 3.1.



Player(**Id** ,Name, Team, **Valid_Since**)

Figura 3.1: Modelo relacional de uma hipotética base de dados contendo informações sobre jogadores de futebol e seu time corrente

A figura 3.1 contém apenas uma entidade com quatro atributos: o identificador do jogador, seu nome, seu time, e um campo indicando o início da validade da informação de time.

Utilizando como base o ER mostrado anteriormente, a figura 3.2 abaixo mostra a tabela da base de dados que representa a entidade da figura 3.1, populada com os dados do jogador Zlatan Ibrahimovic.

	name character varying	team character varying	valid_since timestamp(6) with time zone
1	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03
2	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03

Figura 3.2: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando instantes

Nessa modelagem, apenas um campo temporal, no qual é armazenado um instante de início de validade, é utilizado. Esse tipo de modelagem de dados temporais oferece alguns benefícios. Inserções de novos dados são bastante simples e intuitivas, uma vez que o tempo de validade de um valor vai do instante indicado no campo temporal associado a ele até o instante indicado no próximo registro referente à mesma entidade. Em um modelo em que o término da validade do valor não fosse implícito, uma inserção iria exigir também uma atualização do término da validade do registro anterior. Voltando à figura 3.2, nela são ilustradas as trocas de time do jogador Ibrahimovic. Os valores dos campos indicam que Ibrahimovic foi jogador da Internazionale de 2006 até 2010, quando se transferiu para o Milan. Entretanto, caso se verificasse que essa informação é equivocada, supondo que ele tenha, por exemplo, jogado entre 2009 e 2010 no Barcelona, antes de se transferir para o Milan, bastaria inserir um registro na base de dados, com a data associada de 2009, que automaticamente os intervalos estariam acertados na base de dados. A figura 3.3 mostra como fica a tabela de registro dos jogadores.

	name character varying	team character varying	valid_since timestamp(6) with time zone
1	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03
2	Zlatan Ibrahimovic	Barcelona	2009-01-01 00:00:00-03
3	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03

Figura 3.3: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando instantes, após uma atualização

Outro benefício apresentado por esse tipo de modelo é a facilidade de verificação de validade de atributos temporais. Por exemplo, um jogador não pode ser de dois clubes ao mesmo tempo. Nesse modelo, pode-se garantir a integridade dos dados em relação à essa restrição utilizando uma simples chave primária que combine o identificador do jogador e a data do registro sendo inserido. Um jogador apenas não pode ter dois registros com mesma data e clubes diferentes. Além disso, nesse modelo não existem falhas na linha temporal de um atributo. Esse modelo garante que a base de dados é capaz de fornecer um valor para esse atributo em qualquer ponto no tempo.

Apesar da facilidade para inserir dados, essa alternativa de representação apresenta também alguns problemas. Esses problemas derivam principalmente do fato de que o período de validade de um determinado valor para um atributo temporal não está contido explicitamente dentro do registro que o representa na base de dados. Para determinar esse período é necessária uma análise sobre toda a tabela onde estão os registros para verificar qual o próximo valor, cronologicamente, e assim descobrir onde se encerra o intervalo de validade de um determinado valor. Voltando ao exemplo

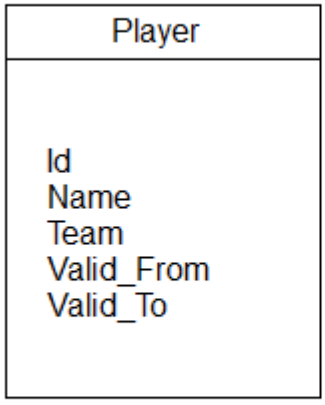
anterior, analisando apenas o primeiro registro da tabela na figura 3.3, é impossível verificar quando termina a validade do valor “Internazionale” para o campo “*team*” na tabela apenas analisando esse registro. É necessário analisar todos os registros da tabela para verificar que essa validade se encerra em 2009, quando o valor “Barcelona” passou a ser o valor válido pra esse campo.

Por esse motivo, consultas que envolvam verificar se algum período arbitrário intersecciona o período de validade de algum registro podem ser complexas de se construir, trazendo possíveis impactos de performance.

3.1.2 Modelo baseado em períodos

Como mencionado anteriormente, usar apenas um campo para representar a temporalidade de atributos de uma base de dados pode trazer alguns complicadores, principalmente associados à dificuldade de determinar o período referente ao tempo de validade do valor de um determinado campo. Uma alternativa a esse modelo de representação é a representação utilizando dois campos de tempo, que representam o período de validade do valor de um atributo temporal. Dessa maneira, para saber a validade do valor de um atributo temporal, basta apenas olhar o próprio registro onde o valor está e verificar seus campos de tempo.

Voltando ao contexto utilizado anteriormente para analisar a aplicação do modelo de instantes, podemos verificar a diferença entre ambos utilizando, no mesmo contexto, a aplicação do modelo baseado em períodos. Na figura 3.4 é mostrado o diagrama ER e a descrição da relação com as alterações para que sejam utilizados dois campos de tempo para determinar o período de validade de cada informação de time.



Player(Id ,Name, Team, Valid_From, Valid_To)

Figura 3.4: Modelo relacional de uma hipotética base de dados contendo informações sobre jogadores de futebol e seu time corrente, agora utilizando períodos para determinar a validade da informação de time

Tomando como base o exemplo usado anteriormente, referente ao jogador Ibrahimovic e suas transferências de clube ao longo do tempo, no modelo de períodos a

tabela que contém os registros de jogadores e seus times teria o estado ilustrado na figura 3.5.

	name character varying	team character varying	valid_from timestamp(6) with time zone	valid_to timestamp(6) with time zone
1	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03	2008-12-31 00:00:00-03
2	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03	2012-06-30 00:00:00-03
3	Zlatan Ibrahimovic	Barcelona	2009-01-01 00:00:00-03	2009-12-31 00:00:00-03

Figura 3.5: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando períodos

Na figura 3.5, podemos verificar que Ibrahimovic foi jogador da Internazionale entre 2006 e 2009. Mesmo sem analisar os outros campos, é possível identificar que o período de validade do valor “Internazionale” para o campo “time” é o período compreendido entre o início de 2006 e o final de 2009. Essa capacidade ilustra o principal ganho apresentado por esse modelo, que é a capacidade de identificar o período de validade de um valor dentro do próprio registro onde ele se encontra. Por conta dessa característica do modelo, consultas que envolvem verificações sobre períodos e intervalos temporais tendem a ser construídas mais facilmente nesse modelo.

Naturalmente, esse modelo também tem suas limitações. A primeira delas se refere ao tratamento que deve ser dado a um registro que não tem o final do seu período de validade definido. Voltando ao exemplo do jogador Ibrahimovic, ele atualmente defende o *Paris Saint-Germain*, clube francês. Entretanto, não se sabe por quanto tempo ele continuará a defender esse mesmo clube. Nesse caso, como deve ser a representação desse fato na base de dados? Como visto anteriormente, o suporte para o conceito de *agora* nos SGBDs comerciais ainda é muito pobre. Algumas abordagens sugerem que um valor nulo seja utilizado para representar essa informação desconhecida. Entretanto, utilizar um valor nulo exige que isso seja considerado nas consultas, que deverão antes de aplicar operações temporais para obtenção de resultados, considerar a existência ou não desse valor nulo. Outra alternativa seria utilizar a data mais longínqua suportada pela base de dados, evitando checagens de valores nulos nas consultas, mas que ainda não resolve todos os problemas. Consultas que tenham por objetivo medir intervalos temporais dos registros e compará-los ainda teriam que estar cientes que o valor escolhido para representar o fim da validade representa o instante atual. O estado da tabela de registro de jogadores, contendo o registro que representa o clube atual do jogador Ibrahimovic é apresentado na figura 3.6.

	name character varying	team character varying	valid_from timestamp(6) with time zone	valid_to timestamp(6) with time zone
1	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03	2008-12-31 00:00:00-03
2	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03	2012-06-30 00:00:00-03
3	Zlatan Ibrahimovic	Barcelona	2009-01-01 00:00:00-03	2009-12-31 00:00:00-03
4	Zlatan Ibrahimovic	Paris Saint-Germain	2012-07-01 00:00:00-03	9999-12-30 00:00:00-03

Figura 3.6: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando períodos, contendo um registro que representa o clube atual do jogador

Outro problema relacionado é referente às inserções sobre uma base de dados que utilize esse modelo. Quando um valor é inserido, ele possivelmente representa uma alteração em dados já existentes. Isso porque muitas vezes um atributo temporal pode ter apenas um valor ao longo do tempo. Voltando ao exemplo anterior, a figura 3.6 mostra a situação do jogador Ibrahimovic atualmente, com a tabela contendo seu clube atual. Conforme mencionado anteriormente, o limite superior do intervalo de validade

referente ao clube atual do jogador contém o valor máximo suportado para um campo de data pelo SGBD. Imaginando uma situação em que o jogador troque de clube novamente, um novo registro com o novo clube do jogador deve ser inserido. Nesse caso, entretanto, o registro contendo o clube que era o atual do jogador deve ser atualizado para conter, no campo que representa o limite superior do período de validade desse registro, a data em que se encerrou o vínculo entre clube e jogador, na qual ele passa a ser jogador do seu novo clube. A figura 3.7 ilustra o estado da tabela após essas alterações.

	name character varying	team character varying	valid_from timestamp(6) with time zone	valid_to timestamp(6) with time zone
1	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03	2008-12-31 00:00:00-03
2	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03	2012-06-30 00:00:00-03
3	Zlatan Ibrahimovic	Barcelona	2009-01-01 00:00:00-03	2009-12-31 00:00:00-03
4	Zlatan Ibrahimovic	Paris Saint-Germain	2012-07-01 00:00:00-03	2014-12-31 00:00:00-03
5	Zlatan Ibrahimovic	Internacional	2015-01-01 00:00:00-03	9999-12-30 00:00:00-03

Figura 3.7: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando períodos, agora com mais uma atualização

Um terceiro problema existente nesse modelo de representação é o fato de que ele permite que existam pontos no tempo em que o atributo temporal modelado não possua nenhum valor. Em contextos nos quais isso não seja permitido, se faz necessário algum tipo de intervenção externa sobre a tabela que contém o atributo pra garantir a integridade dos dados. Normalmente, funções de gatilho (*triggers*) são utilizadas pra garantir essa integridade dos dados. Da mesma maneira, deve-se evitar que um atributo temporal assuma dois valores em um mesmo instante. Para isso, se faz necessária a presença de um procedimento externo pra garantir a integridade dos dados. Apenas chaves primárias não são suficientes para evitar esse tipo de ocorrência. Essas dependências de funções externas geradas por essas características do modelo podem afetar o desempenho das consultas sobre a base de dados, assim como aumentar a simplicidade da criação e manutenção da mesma.

3.1.3 Tabelas de histórico e tabelas atemporais

Independentemente do modelo escolhido, é possível separar os atributos temporais e atemporais da entidade, de forma a otimizar o uso da base de dados (GUNADHI , 1989; ESRI, 2012). Dessa forma, é possível simplificar consultas que não envolvem informações temporais, além de manter constantemente uma imagem do estado atual da entidade representada. Além disso, é possível economizar o espaço em disco utilizado pela base com esse tipo de separação.

Usando novamente o exemplo utilizado nas sessões anteriores, podemos aplicar essa alternativa na tabela de registro de jogadores. Caso fosse necessário armazenar, para cada jogador, também informações pessoais como números de CPF e RG, essas informações que não variam no tempo seriam desnecessariamente replicadas para cada variação do atributo temporal. Nesse caso, pode-se fazer uso de uma tabela separada para armazenar essas informações, enquanto que em uma segunda tabela ficariam apenas o identificador da entidade, o valor do atributo temporal, e o(s) campo(s) referente ao tempo de validade do atributo. A figura 3.8 mostra como ficam as entidades e o relacionamento entre elas no modelo relacional.

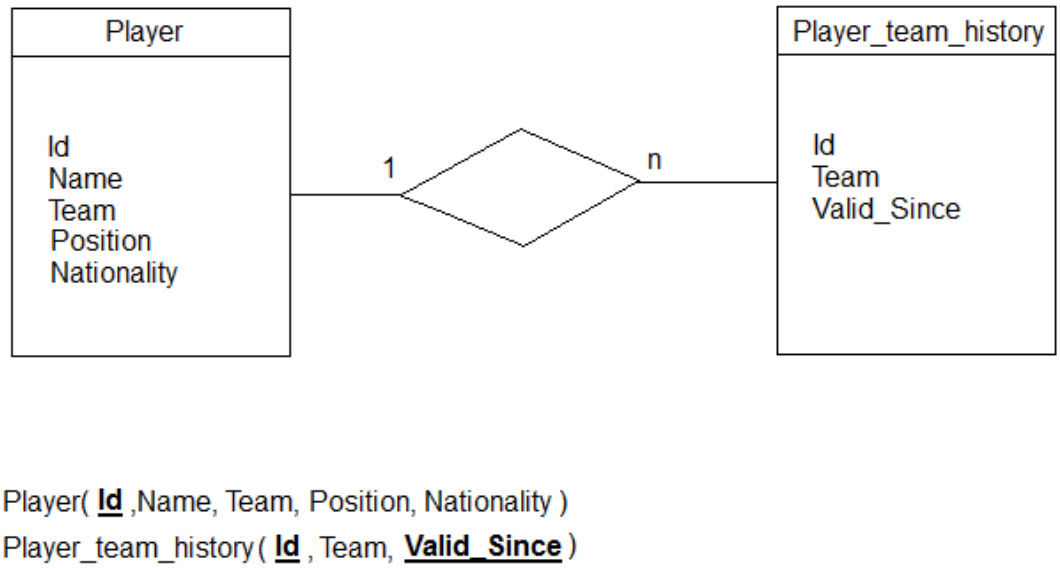


Figura 3.8: Modelo relacional de uma hipotética base de dados contendo informações sobre jogadores de futebol e seu time corrente, agora utilizando uma tabela a parte para armazenar os dados temporais

Como é possível verificar na Figura 3.8, a tabela com os dados atemporais também contém um campo referente ao time do jogador. Esse campo é utilizado para armazenar o valor corrente do atributo "*team*" representado na tabela separada. Essa é uma otimização muitas vezes desejável para acelerar consultas de estado corrente, e nesse trabalho se optou por utilizá-la nas modelagens de atributos temporais que envolvem tabelas separadas. Naturalmente, isso requer uma atualização a mais sempre que o time atual do jogador for atualizado, sendo mais uma questão a ser estudada.

Nas figuras 3.9 e 3.10 é mostrado o estado das tabelas em um dos modelos anteriores e nesse modelo com uma tabela à parte para armazenar dados temporais, onde estão representadas as relações mostradas na figura 3.8.

	id integer	name character varying	team character varying	valid_since timestamp(6) with time zone	position character varying	nationality character varying
1	80000	Zlatan Ibrahimovic	Internazionale	2006-01-01 00:00:00-03	Striker	Swedish
2	80000	Zlatan Ibrahimovic	Barcelona	2009-01-01 00:00:00-03	Striker	Swedish
3	80000	Zlatan Ibrahimovic	Milan	2010-01-01 00:00:00-03	Striker	Swedish

Figura 3.9: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando instantes, contendo campos não temporais

Novamente utilizando o jogador Ibrahimovic como exemplo, a figura 3.9 mostra como fica a tabela representando o histórico de times do jogador, contendo também e suas informações que não variam no tempo. Como apenas uma tabela é utilizada para armazenar os dados, sempre que o atributo temporal for atualizado, o que pode ser causado por uma troca de time do jogador, por exemplo, todos os outros dados da tabela precisam ser copiados e estarem presentes no registro que conterá esse novo valor do atributo temporal. Quando a idade do jogador for atualizada, não apenas um registro necessitará de atualização na tabela, mas vários.

	id integer	name character varying	team character varying	position character varying	nationality character varying
1	80000	Zlatan Ibrahimovic	Paris Saint-Germain	Striker	Swedish

	id integer	team character varying	valid_since timestamp with time zone
1	80000	Internazionale	2006-01-01 00:00:00-03
2	80000	Barcelona	2009-01-01 00:00:00-03
3	80000	Milan	2010-01-01 00:00:00-03
4	80000	Paris Saint-Germain	2012-07-01 00:00:00-03
5	80000	Internacional	2015-01-01 00:00:00-03

Figura 3.10: Informações sobre o jogador Zlatan Ibrahimovic em um modelo de representação temporal utilizando instantes, contendo campos não temporais, mas separados em uma segunda tabela

Já na figura 3.10, os dados temporais e atemporais são separados. Nessa representação, os dados atemporais ficam separados em uma tabela, enquanto o atributo temporal e seu histórico ficam em outra. É utilizado o atributo “identificador” para fazer a conexão entre as tabelas. Assim, consultas que não envolvem o atributo temporal, como por exemplo, uma consulta para obter a nacionalidade do jogador Ibrahimovic, são beneficiadas pelo fato de que elas precisarão percorrer uma tabela que tende a ser menos densa do que seria se o atributo temporal não fosse separado. Além disso, voltando ao caso da atualização da idade do jogador, apenas um único campo seria alterado, e não todos os registros de histórico existentes na base referentes ao jogador.

Essa abordagem tende a otimizar o espaço em disco utilizado para a representação de entidades com atributos temporais em uma base de dados e a melhorar o desempenho de consultas atemporais. Entretanto, o fato de consultas de histórico necessitarem agora de uma junção de tabelas pode trazer consequências para o desempenho de determinadas consultas, além de trazer uma simplicidade adicional para a construção das mesmas. Intuitivamente é complicado medir qual a proporção entre ganho de espaço e detrimento de desempenho dessa abordagem. Isso depende de alguns fatores, como o numero de campos atemporais da entidade, e também da frequência com que consultas temporais são executadas no contexto em que está inserida a base de dados, sendo necessários experimentos para quantificar essa proporção.

3.2 Entidades temporais

Em algumas aplicações de bases de dados, representar apenas um atributo como temporal não é uma alternativa possível ou capaz de atender à demanda das aplicações que consomem dados da base de dados, uma vez que se faz necessário conhecer os estados da entidade como um todo ao longo do tempo, e não apenas de alguns atributos. Essas entidades são chamadas entidades temporais.

Existem algumas maneiras de lidar com entidades temporais, registrando seus estados ao longo do tempo. Elas são basicamente as mesmas utilizadas para lidar com atributos temporais, porém o tratamento dado aos campos que representam o período de validade da entidade é um pouco diferente. Os campos temporais de cada registro se referem a entidade como um todo, uma vez que em uma entidade temporal todos os atributos são temporais. As vantagens e desvantagens de cada modelo, seja utilizando instantes ou períodos, são similares às vantagens e desvantagens apresentadas pelos

mesmos modelos quando referentes a atributos temporais. Existem, entretanto, alguns problemas adicionais quando se lida com entidades temporais.

O principal problema encontrado nessa situação é referente à quando existem referências de chave estrangeira para uma tabela que representa uma entidade temporal (SNODGRASS, 1999-a; TANSEL, 2004). Para exemplificar essa situação, podemos utilizar o contexto de um jogo em que existem duas entidades temporais, uma chamada “personagem”, e outra chamada “classe de personagem”. Na figura 3.11, o modelo ER referente a essas entidades.

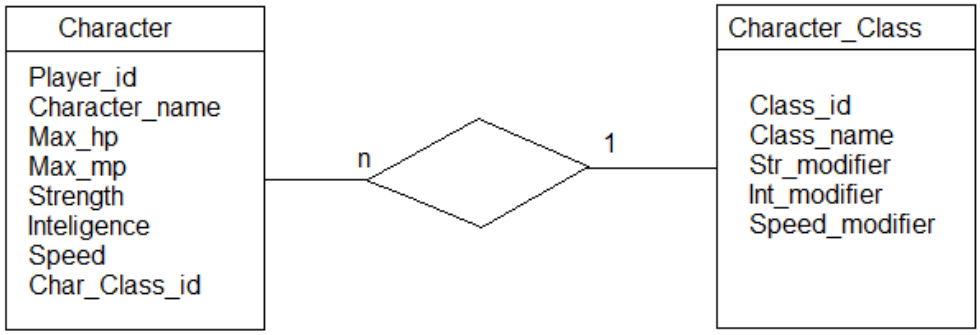


Figura 3.11: Modelo ER representando as entidades “personagem” e “classe de personagem” e a relação entre elas

O personagem representa um jogador e seus atributos no jogo, como por exemplo, “força”, “inteligência” e “velocidade”. A “classe de personagem” é referente à profissão ou especialização do personagem dentro do jogo, como por exemplo, “mago”, “guerreiro” ou “mercador”. Para esse jogo específico, ambas as entidades são temporais, e sempre que algum atributo de alguma delas for alterado, é criado um novo registro temporal na tabela que às representa. Todo e qualquer personagem precisa ter uma classe, por isso existe uma relação de chave estrangeira entre elas. As tabelas, sem os campos que indicam a validade dos dados no tempo, são mostradas na figura 3.12.

	player_id integer	character_name character varying	max_hp integer	max_mp integer	level integer	strength integer	inteligence integer	speed integer	char_class_id integer
1	40000	Crono	311	44	5	30	12	31	60
2	40001	Lucca	263	102	5	14	27	15	61

	class_id integer	class_name character varying	str_modifier integer	int_modifier integer	speed_modifier integer
1	60	Warrior	10	-3	5
2	61	Mage	-5	15	0
3	62	Merchant	5	8	3

Figura 3.12: Tabelas de classes e personagens, sem atributos temporais

A figura 3.12 mostra as duas tabelas representando as entidades e seus atributos sem campos temporais, pois esses serão discutidos mais tarde. Na figura, pode-se identificar

o campo que faz a ligação entre “personagem” e “classe”, o campo “*char_class_id*” na tabela “personagem”, que corresponde ao campo “id” na tabela de classes. Numa aplicação atemporal convencional, para estabelecer essa relação e garantir a integridade da mesma na base de dados, basta apenas utilizar o mecanismo de chaves estrangeiras oferecido pelo SGBD e o problema estaria resolvido. Isso porque em uma aplicação atemporal, basta garantir que, para todos os registros da tabela que faz referência, existe um registro na tabela referenciada que satisfaz a restrição de chave estrangeira. Entretanto, quando o relacionamento envolve entidades temporais, é feito um adendo à essa definição. É necessário garantir que, para todos os registros da tabela que faz referência, existe um registro na tabela referenciada que satisfaz a restrição de chave estrangeira *em todos os pontos no tempo.* O modelo de representação escolhido pode não garantir, por si só, essa restrição, e podem haver pontos no tempo em que a tabela referenciada não oferece valor algum para algum ponto no tempo em que a entidade que faz referencia existe. Para exemplificar melhor essa situação, na figura 3.13 são apresentados alguns exemplos de registros nas tabelas do exemplo abordado.

	player_id integer	character_name character varying	max_hp integer	max_mp integer	level integer	strength integer	inteligence integer	speed integer	valid_from timestamp without time zone	valid_to timestamp without time zone	char_class_id integer
1	40000	Crono	311	44	5	30	12	31	1999-01-01 00:00:00	2001-12-31 00:00:00	60
2	40000	Crono	405	60	6	41	14	45	2002-01-01 00:00:00	2006-06-10 00:00:00	60

	class_id integer	class_name character varying	str_modifier integer	int_modifier integer	speed_modifier integer	valid_from timestamp without time zone	valid_to timestamp without time zone
1	60	Warrior	10	-3	5	1998-01-01 00:00:00	2002-12-31 00:00:00
2	60	Warrior	12	-5	5	2003-04-24 00:00:00	2010-12-31 00:00:00

Figura 3.13: Registros referentes ao personagem Crono e sua classe de personagem ao longo do tempo

Na figura 3.13, existem alguns registros que mostram diferentes valores para as entidades do exemplo no tempo. Nessa figura, são utilizados períodos para expressar o tempo de validade de cada registro. Além disso, nesse exemplo a granularidade temporal utilizada é o dia (apesar dos campos serem do tipo *timestamp*). A primeira linha da tabela de personagens mostra os valores dos atributos do personagem Crono no período que compreende o primeiro dia do ano 1999 até o ultimo dia do ano de 2001. A segunda compreende os valores dos mesmos atributos do primeiro dia de 2002 até o dia 10 de junho de 2006. De maneira similar, a tabela de classes possui dois registros, um compreendendo os atributos de classe da classe Warrior válidos do início de 1998 até o final de 2002, e outro compreendendo outros valores dos mesmos atributos, válidos de 24 de abril de 2003 até o final do ano de 2010. Conforme mencionado antes, o campo “*char_class_id*” indica que Crono é da classe Warrior, e existe uma relação de chave estrangeira entre o valor desse campo e os valores encontrados no campo “*class_id*” da tabela classe. Logo, uma consulta cujo objetivo é recuperar os atributos da classe do personagem Crono em 28 de junho de 2005 irá recuperar os atributos contidos no segundo registro da tabela de classes na figura 3.13. Porém, qual será o resultado para a mesma consulta, trocando apenas a data da mesma para 3 de janeiro de 2003? Conforme mostrado na figura 3.14, não existem dados para a classe guerreiro no período compreendido entre o início do ano de 2003 e o dia 23 de abril do mesmo ano.

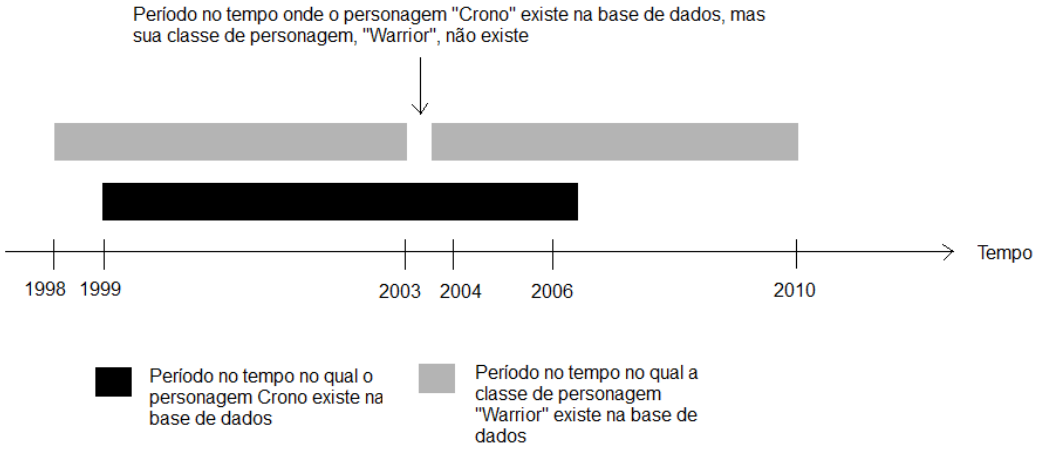


Figura 3.14: Gráfico exemplificando uma violação da restrição de integridade referente a chaves estrangeiras temporais

Logo, qualquer consulta referente a esse período, como a consulta mencionada anteriormente, não seria capaz de retornar nenhuma informação de classe para o personagem Crono, o que constitui uma violação na restrição de integridade existente entre as duas tabelas. Garantir essa restrição de integridade ao longo do tempo é um problema a mais a ser abordado quando é feita a decisão de qual modelo de representação utilizar para os dados temporais. Os modelos propostos são basicamente os mesmos utilizados para representar atributos temporais, mas os problemas adicionais gerados pela necessidade de garantir a integridade dos dados geram outros aspectos a serem analisados em cada modelo, diferente dos abordados em relação à utilização dos mesmos modelos para modelar atributos temporais.

3.2.1 Modelo de representação baseado em instantes

Da mesma maneira como é feito com atributos, instantes podem ser utilizados para definir o tempo de validade dos registros de uma entidade. Usa-se apenas um campo com informação de tempo, que representa o início da validade da tupla. O modelo é similar ao utilizado com atributos, com a diferença que o campo temporal se refere à entidade como um todo e não apenas a um atributo específico. Da mesma forma, o registro referente à entidade tem validade até o início da validade do próximo registro, em uma escala cronológica. No caso do exemplo apresentado anteriormente acerca de personagens e classes, os valores dos instantes ficariam conforme apresentados na figura 3.15 abaixo.

	player_id integer	character_name character varying	max_hp integer	max_mp integer	level integer	strength integer	inteligence integer	speed integer	valid_since timestamp without time zone	char_class_id integer
1	40000	Crono	311	44	5	30	12	31	1999-01-01 00:00:00	60
2	40000	Crono	405	60	6	41	14	45	2002-01-01 00:00:00	60

	class_id integer	class_name character varying	str_modifier integer	int_modifier integer	speed_modifier integer	valid_since timestamp without time zone
1	60	Warrior	10	-3	5	1998-01-01 00:00:00
2	60	Warrior	12	-5	5	2003-04-24 00:00:00

Figura 3.15: Dados referentes ao personagem Crono, em um modelo baseado em instantes

Diferentemente, porém, do exemplo apresentado anteriormente, nesse modelo não são permitidas falhas temporais na base de dados. A partir do momento que uma entidade passa a existir na base de dados, ela sempre terá valores definidos no tempo, começando no instante associado ao seu primeiro registro. Lidando com atributos essa característica não chega a ser muito importante, uma vez que se um atributo não tiver valor em um determinado ponto no tempo pode-se simplesmente assumir um valor nulo para o atributo naquele instante. Já lidando com entidades, o problema apresentado no exemplo anterior seria mais difícil de acontecer: o personagem Crono teria valores definidos para sua classe em qualquer ponto no tempo a partir do início de sua existência. Isso afeta diretamente a maneira como o código para checagem de restrições de integridade é escrito, sendo o mesmo mais simples e tendendo a ter desempenho muito melhor.

Naturalmente, da mesma maneira que o modelo análogo aplicável a atributos, esse modelo apresenta menor simplicidade para se escrever consultas sobre intervalos temporais. As limitações e facilitadores desse modelo serão analisados mais a fundo no próximo capítulo.

3.2.2 Modelo de representação baseado em períodos

O modelo de dados baseado em períodos também se aplica para entidades. Esse modelo possui as mesmas características do modelo análogo aplicado para atributos, com a diferença que os campos temporais agora se referem a entidade como um todo. Utilizam-se dois campos com informação de tempo, que representam as extremidades do intervalo de validade de cada tupla. Voltando ao exemplo apresentado no início da seção, nesse modelo de representação os campos ficariam conforme apresentados na figura 3.16 abaixo.

	player_id integer	character_name character varying	max_hp integer	max_mp integer	level integer	strength integer	inteligence integer	speed integer	valid_from timestamp without time zone	valid_to timestamp without time zone	char_class_id integer
1	40000	Crono	311	44	5	30	12	31	1999-01-01 00:00:00	2001-12-31 00:00:00	60
2	40000	Crono	405	60	6	41	14	45	2002-01-01 00:00:00	2006-06-10 00:00:00	60

	class_id integer	class_name character varying	str_modifier integer	int_modifier integer	speed_modifier integer	valid_from timestamp without time zone	valid_to timestamp without time zone
1	60	Warrior	10	-3	5	1998-01-01 00:00:00	2002-12-31 00:00:00
2	60	Warrior	12	-5	5	2003-04-24 00:00:00	2010-12-31 00:00:00

Figura 3.16: Dados referentes ao personagem Crono, em um modelo baseado em períodos

Esse modelo, diferentemente do modelo baseado em instantes, permite que ocorra o problema das falhas temporais mencionado anteriormente. Esse fato faz com que o código externo responsável por manter a integridade dos dados na base necessite ser mais complexo e possivelmente tenha desempenho pior. Assim como para o modelo baseado em instantes, no capítulo seguinte serão analisados em detalhe os efeitos das características desse modelo em relação a entidades.

3.3 Registro de alterações de tempo de transação

Com frequência, acontece de a necessidade de se manter um histórico dos dados existentes na base de dados surgir após a criação e implantação de um sistema. Além disso, muitas vezes a principal motivação para o armazenamento do histórico de uma base de dados é a possibilidade de recuperação do estado da mesma em uma data

anterior, devido a alguma perda de dados ou evento similar, não sendo a aplicação temporal por natureza. Nessas situações, é indicada a utilização de um *registro de alterações em tempo de transação* (SNODGRASS, 1999-b; ALLEN, 2000). Esse registro consiste na criação de tabelas auxiliares que contém um registro de alterações realizadas nas tabelas do sistema para posterior uso para consultas e recuperação de estados anteriores.

Um sistema com essas características utiliza os conceitos de *tabela monitorada* e *tabela de monitoramento*. A tabela monitorada é a tabela de fato utilizada pelo sistema, enquanto a tabela de monitoramento é a tabela onde são armazenadas as alterações realizadas sobre a tabela monitorada.

Para exemplificar a aplicação desse conceito sobre uma base de dados, será utilizado novamente o contexto de um jogo onde cada personagem tem uma série de atributos. A principal tabela do jogo modelado seria a tabela de personagens (*game_character*), similar à tabela usada na seção anterior. Essa tabela é monitorada e as alterações causadas nela são guardadas na tabela de monitoramento *c_log*. A descrição de ambas as tabelas no modelo relacional é mostrada na figura 3.17, enquanto as tabelas em si são mostradas na figura 3.18 abaixo.

Game_Character (**Character_id**, Character_name, Speed, Strength, Inteligence, Char_Class_Name)
C_Log (**Character_id**, Character_name, Speed, Strength, Inteligence, Char_Class_Name, When_Changed)

Figura 3.17: Tabelas monitorada e de monitoramento no modelo relacional

Tabela c_log

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	when_changed timestamp without time zone	char_class_name character varying
1	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	Knight
2	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	Knight
3	100200	Edgar	5	8	3	2	2010-01-06 00:00:00	Knight
4	100200	Edgar	6	10	4	3	2010-01-07 00:00:00	Knight
5	100200	Edgar	8	13	5	4	2010-01-09 00:00:00	Knight
6	100200	Edgar	9	15	6	5	2010-01-10 00:00:00	Knight
7	100200	Edgar	11	18	7	6	2010-01-12 00:00:00	Knight
8	100200	Edgar	12	20	8	6	2010-01-13 00:00:00	Knight
9	100200	Edgar	14	23	9	7	2010-01-15 00:00:00	Knight
10	100200	Edgar	15	25	10	8	2010-01-16 00:00:00	Knight
11	100200	Edgar	17	28	11	9	2010-01-18 00:00:00	Knight
12	100200	Edgar	18	30	12	10	2010-01-19 00:00:00	Knight
13	100200	Edgar	20	33	13	10	2010-01-21 00:00:00	Knight
14	100200	Edgar	21	35	14	11	2010-01-22 00:00:00	Knight
15	100200	Edgar	23	38	15	12	2010-01-24 00:00:00	Knight
16	100200	Edgar	24	40	16	13	2010-01-25 00:00:00	Knight
17	100200	Edgar	26	43	17	14	2010-01-27 00:00:00	Knight
18	100200	Edgar	27	45	18	14	2010-01-28 00:00:00	Knight
19	100200	Edgar	29	48	19	15	2010-01-30 00:00:00	Knight
20	100200	Edgar	30	70	20	26	2010-01-31 00:00:00	Royal Guard
21	100200	Edgar	32	74	21	27	2010-02-02 00:00:00	Royal Guard
22	100200	Edgar	33	77	22	29	2010-02-03 00:00:00	Royal Guard
23	100200	Edgar	35	81	23	30	2010-02-05 00:00:00	Royal Guard
24	100200	Edgar	36	84	24	31	2010-02-06 00:00:00	Royal Guard
25	100200	Edgar	38	88	25	33	2010-02-08 00:00:00	Royal Guard
26	100200	Edgar	39	91	26	34	2010-02-09 00:00:00	Royal Guard

Tabela game_character

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying
1	100200	Edgar	41	95	27	35	Royal Guard

Figura 3.18: Tabela monitorada com valores atuais para o personagem “Edgar” e tabela de monitoramento com histórico de valores para o mesmo personagem

A figura 3.18 ilustra a tabela monitorada e a tabela de monitoramento. Sempre que alguma alteração acontece na tabela monitorada, um registro correspondente é criado na tabela de monitoramento, contendo a informação temporal de alteração. Pelos registros da tabela de monitoramento, é possível deduzir que o valor de força do personagem Edgar sofreu alterações ao longo do tempo, assim como sua classe, que mudou de “*Knight*” para “*Royal Guard*”.

Existem algumas alternativas para modelagem desse registro de alterações em tempo de transação. A alternativa de modelagem escolhida afeta a maneira como os procedimentos de reconstrução e as consultas são escritos. Existem quatro alternativas principais de modelagem, que serão estudadas em detalhe nas próximas sessões.

3.3.1 Modelo sem inserções

Partindo de um pressuposto em que novos dados não são inseridos na tabela, apenas os dados existentes alterados, é possível utilizar um modelo em que apenas atualizações e remoções são registradas na tabela de monitoramento. Como não existem inserções de novos dados, apenas isso é suficiente para reconstruir a tabela monitorada em qualquer ponto do tempo. Esse modelo tem pouca aplicação prática, mas serve de base para os modelos que serão apresentados mais adiante.

Nesse modelo, qualquer alteração (atualizações ou remoções) executada sobre a tabela monitorada é guardada na tabela de monitoramento. Na figura 3.19 abaixo é possível verificar esse comportamento.

Tabela c_log

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	when_changed timestamp without time zone	char_class_name character varying
1	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	Knight
2	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	Knight
3	100200	Edgar	5	8	3	2	2010-01-06 00:00:00	Knight
4	100200	Edgar	6	10	4	3	2010-01-07 00:00:00	Knight
5	100200	Edgar	8	13	5	4	2010-01-09 00:00:00	Knight

Figura 3.19: Histórico do personagem “Edgar” na tabela de monitoramento

Na figura 3.19 podemos ver os valores do campo “*Strength*” da tabela monitorada que se perderam por conta de atualizações. Nesse modelo são armazenados na tabela de monitoramento os valores anteriores das tuplas. Ou seja, se usam *imagens anteriores* da tupla na tabela de monitoramento. No exemplo da figura, podemos verificar que o valor de força do personagem Edgar passou de 10 para 13 no dia 7 de Janeiro de 2010. Além do valor anterior da tupla, é armazenada também a data da alteração. Esse campo permite que sejam executadas consultas temporais sobre a tabela de monitoramento. Podem-se adicionar ainda outros campos na tabela de monitoramento, como por exemplo, o nome do usuário que fez a alteração ou algo similar, para investigações posteriores.

As inserções na tabela de monitoramento são feitas automaticamente. É possível programar gatilhos (*triggers*) sobre a tabela monitorada, para que cada atualização ou remoção sobre ela gere um registro na tabela de monitoramento. Assim, usando o exemplo da figura 3.19, no momento em que a tabela monitorada sofre uma atualização em quaisquer campos, antes dessa alteração ser finalizada os valores antigos dos campos da tupla são armazenados na tabela de monitoramento. Essa operação deve ser atômica, ou seja, a atualização só funciona se a inserção na tabela de monitoramento funcionar. O funcionamento para remoções de dados é similar. Quando uma tupla é removida, seu conteúdo é inserido na tabela de monitoramento antes da remoção terminar.

Com a tabela de monitoramento, certas ações são possíveis para recuperar e consultar dados históricos da tabela monitorada. É possível reconstruir a tabela monitorada em um dado momento no tempo, traduzir a tabela de monitoramento em uma tabela de tempo de validade para consultas temporais mais amplas e ainda é possível selecionar dados históricos diretamente na tabela de monitoramento.

Recuperar a tabela monitorada para um ponto anterior no tempo pode ser útil para desfazer alterações indesejadas, ou para análises que necessitem retroceder os dados para um determinado ponto no tempo. A partir da tabela de monitoramento é possível criar uma visão (*view*) no banco de dados representando a tabela como ela era no ponto desejado. Para tanto basta fazer um *select* sobre a tabela monitorada colocando o resultado desse *select* na *view*. A figura 3.20 exemplifica isso.

```
1  create view game_characters_as_in_october_5th_2009 AS
2  SELECT * FROM game_character   AS P WHERE NOT EXISTS (SELECT * FROM c_log_ AS A WHERE P.character_id = A.
   character_id AND A.When_Changed > DATE '2009-10-05')
3  UNION
4  SELECT character_id, character_name, speed, strength, character_level, inteligence , character_class FROM
   c_log_ AS A WHERE When_Changed = (SELECT MIN(When_Changed) FROM c_log_ AS A2 WHERE A.character_id = A2.
   character_id AND A2.When_Changed > DATE '2009-10-05')
```

game_characters_as_in_october_5th_2009

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying
1	100200	Edgar	2	3	1	1	Knight

c_log

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	when_changed timestamp without time zone	char_class_name character varying
1	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	Knight
2	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	Knight
3	100200	Edgar	5	8	3	2	2010-01-06 00:00:00	Knight
4	100200	Edgar	6	10	4	3	2010-01-07 00:00:00	Knight
5	100200	Edgar	8	13	5	4	2010-01-09 00:00:00	Knight

Figura 3.20: Consulta que gera a *view* que representa a tabela monitorada no dia 5 de Outubro de 2009, a tabela gerada, e o estado da tabela de monitoramento

Na figura 3.20 é mostrada a consulta que gera a visão da tabela monitorada como ela era na data alvo, que no caso da figura, é a data 5 de outubro de 2009. Na consulta da figura, essa visão recebe o nome de *game_characters_as_in_october_5th*. O resultado é uma visão com os dados da tabela monitorada como eles eram nesse dia. Essa consulta busca primeiro, através de uma subconsulta, registros na tabela monitorada que não foram alterados depois da data alvo, ou seja, para o caso de o registro na tabela monitorada ser o registro válido na data alvo (linha 1). Os registros dessa primeira subconsulta são unidos com registros da tabela de monitoramento, para dados que foram alterados depois da data alvo. O valor desses registros estará contido na tabela de monitoramento, e o seu *timestamp* relacionado (campo *When_Changed*) será o menor valor que seja maior que a data alvo (linha 2).

No exemplo em questão, o resultado dessa consulta é o registro referente ao personagem “Edgar” originalmente inserido na tabela, já que as modificações só acontecem a partir do início de 2010, como mostrado na tabela de monitoramento *c_log*.

É possível também consolidar todas as informações da tabela de monitoramento numa tabela de tempo de validade, utilizando períodos, de modo que todas as informações tenham valores de início e fim de validade na tabela resultante. Assim é possível executar consultas temporais mais facilmente. Assim como ao reconstruir a tabela monitorada em uma data específica, é possível realizar essa operação através de uma consulta cujo resultado é colocado numa visão. A consulta que gera a tabela de tempo de validade, assim como um exemplo de resultado dessa consulta, é mostrada na figura 3.21.

```
1SELECT character_id, character_name, speed,strength, character_level, inteligence,timestamp '2009-10-01
 00:00:00' as valid_from, current_timestamp  as valid_to FROM game_characters_withoutinserts WHERE
2NOT EXISTS ( SELECT * FROM c_log_withoutInserts WHERE c_log_withoutInserts.character_id =
 game_characters_withoutinserts.character_id)
3UNION
4SELECT character_id, character_name, speed, strength, character_level, inteligence,timestamp '2009-10-01
 00:00:00' as valid_from, When_Changed as valid_to FROM c_log_withoutInserts
5    AS A1 WHERE NOT EXISTS  ( SELECT * FROM c_log_withoutInserts AS A2 WHERE A1.character_id = A2.
     character_id AND A1.When_Changed > A2.When_Changed)
6UNION
7 SELECT A1.character_id, A1.character_name, A1.speed,A1.strength, A1.character_level, A1.inteligence,
8 When_Changed as valid_from , current_timestamp as valid_to FROM c_log_withoutInserts
9 AS A1, game_characters_withoutinserts WHERE A1.character_id = game_characters_withoutinserts.character_id
10 AND NOT EXISTS  ( SELECT * FROM c_log_withoutInserts AS A2 WHERE A1.character_id = A2.character_id AND A1.
   When_Changed < A2.When_Changed)
11UNION
12 SELECT A1.character_id, A1.character_name, A1.speed, A1.strength, A1.character_level, A1.inteligence, A0.
   When_Changed as valid_from , A1.When_Changed as valid_to FROM c_log_withoutInserts AS A0,
   c_log_withoutInserts AS A1
13 WHERE A0.character_id = A1.character_id AND A0.When_Changed < A1.When_Changed AND NOT EXISTS (
14 SELECT * FROM c_log_withoutInserts AS M WHERE M.character_id = A1.character_id AND M.When_Changed < A1.
   When_Changed AND M.When_Changed > A0.When_Changed)
```

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	valid_from timestamp	valid_to timestamp
1	100200	Edgar	2	3	1	1	2009-10-01 00:00:00	2010-01-03 00:00:00
2	100200	Edgar	3	5	2	2	2010-01-03 00:00:00	2010-01-04 00:00:00
3	100200	Edgar	5	8	3	2	2010-01-04 00:00:00	2010-01-06 00:00:00
4	100200	Edgar	6	10	4	3	2010-01-06 00:00:00	2010-01-07 00:00:00
5	100200	Edgar	8	13	5	4	2010-01-07 00:00:00	2010-01-09 00:00:00
6	100200	Edgar	9	15	6	5	2010-01-09 00:00:00	2010-01-10 00:00:00
7	100200	Edgar	11	18	7	6	2010-01-10 00:00:00	2010-01-12 00:00:00
8	100200	Edgar	12	20	8	6	2010-01-12 00:00:00	2010-01-13 00:00:00
9	100200	Edgar	14	23	9	7	2010-01-13 00:00:00	2010-01-15 00:00:00
10	100200	Edgar	15	25	10	8	2010-01-15 00:00:00	2010-01-16 00:00:00
11	100200	Edgar	17	28	11	9	2010-01-16 00:00:00	2010-01-18 00:00:00
12	100200	Edgar	18	30	12	10	2010-01-18 00:00:00	2010-01-19 00:00:00
13	100200	Edgar	20	33	13	10	2010-01-19 00:00:00	2010-01-21 00:00:00
14	100200	Edgar	21	35	14	11	2010-01-21 00:00:00	2010-01-22 00:00:00
15	100200	Edgar	23	38	15	12	2010-01-22 00:00:00	2010-01-24 00:00:00
16	100200	Edgar	24	40	16	13	2010-01-24 00:00:00	2010-01-25 00:00:00
17	100200	Edgar	26	43	17	14	2010-01-25 00:00:00	2010-01-27 00:00:00
18	100200	Edgar	27	45	18	14	2010-01-27 00:00:00	2010-01-28 00:00:00
19	100200	Edgar	29	48	19	15	2010-01-28 00:00:00	2010-01-30 00:00:00
20	100200	Edgar	30	70	20	26	2010-01-30 00:00:00	2010-01-31 00:00:00
21	100200	Edgar	32	74	21	27	2010-01-31 00:00:00	2010-02-02 00:00:00
22	100200	Edgar	33	77	22	29	2010-02-02 00:00:00	2010-02-03 00:00:00
23	100200	Edgar	35	81	23	30	2010-02-03 00:00:00	2010-02-05 00:00:00
24	100200	Edgar	36	84	24	31	2010-02-05 00:00:00	2010-02-06 00:00:00
25	100200	Edgar	38	88	25	33	2010-02-06 00:00:00	2010-02-08 00:00:00
26	100200	Edgar	39	91	26	34	2010-02-08 00:00:00	2010-02-09 00:00:00
27	100200	Edgar	39	91	26	34	2010-02-09 00:00:00	2012-09-30 19:30:41

Figura 3.21: Consulta para gerar uma tabela com períodos de início e fim de validade, através dos dados das tabelas monitoradas e de monitoramento

A consulta na figura 3.21 monta uma tabela com dados de início e fim de validade para cada registro. Nessa consulta, tanto a tabela de monitoramento quanto a tabela monitorada receberam o sufixo "*_without_inserts*" para identificar à qual modelo se referem.

Os dados iniciais, resultantes das primeiras inserções na tabela monitorada, são inseridos no resultado com a informação de início de validade igual ao ponto no tempo em que a tabela foi definida, no caso do exemplo, 1° de Outubro de 2009 (linhas 1 a 5 da consulta na figura). Isso ocorre devido ao fato que a tabela de monitoramento não recorda inserções. A consulta das linhas 1 e 2 busca dados que foram inseridos e não foram mais alterados. Já a consulta das linhas 4 e 5 busca o dado mais antigo referente a uma entidade, no caso desse exemplo, um personagem, e retorna o registro mais antigo referente aquela entidade, com o tempo de validade final igual ao ponto no tempo onde

a entidade foi alterada pela primeira vez. A consulta das linhas 7 a 10 procura pela alteração mais recente sobre a entidade, e define seu tempo de validade final como sendo o instante de agora (*current_timestamp*, linha 8). Por fim, a última subconsulta (linhas 12 a 14) busca pelas alterações intermediárias no tempo sobre a entidade, e monta os períodos temporais de acordo. Essa transformação facilita a visualização do tempo de validade das informações, além de tornar mais fácil a escrita de consultas temporais sobre os dados.

Por fim, é possível consultar diretamente a tabela de monitoramento por uma informação específica no tempo. Muitas vezes não compensa criar uma tabela de tempo de validade ou reconstruir a tabela monitorada em um ponto específico para fazer consultas temporais. Nesse caso, consultas são realizadas diretamente sobre a tabela de monitoramento. Essas consultas são parecidas com as consultas que geram o estado da tabela monitorada em um ponto específico, apenas com a diferença que parâmetros adicionais são adicionados para obter informações mais ou menos específicas. Naturalmente, a consulta que reconstrói a tabela inteira em um ponto do tempo é uma consulta desse tipo, mas também é possível realizar consultas mais específicas para obter determinadas informações diretamente. Na figura 3.22 é possível visualizar um exemplo de uma consulta assim.

```
1  SELECT character_id, character_name, speed, strength, character_level,
2         inteligence, character_class
3    FROM c_log_withoutinserts A where A.character_id  = 100200
4    and
5    A.when_changed > timestamp '11-01-2010' and not exists (SELECT *
6    FROM c_log_withoutinserts B where B.character_id  = 100200
7    and
8    B.when_changed > timestamp '11-01-2010'
9    and
10   B.when_changed < a.when_changed)
11 UNION
12 SELECT *
13   FROM game_characters_withoutinserts where character_id = 100200 AND NOT EXISTS (
14   SELECT * from c_log_withoutinserts
15   where character_id = 100200 and when_changed > timestamp '11-01-2010'  )
```

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	when_changed timestamp without time zone	character_class character varying
1	100200	Edgar	11	18	7	6	2010-01-12 00:00:00	Knight

Figura 3.22: Consulta diretamente sobre o *registro* de alterações e seu resultado

A consulta da figura 3.22 busca o estado do personagem Edgar, cujo id é “100200”, no dia 11 de Janeiro de 2010. Para obter esse resultado, é necessário verificar a primeira alteração depois do dia 11 na tabela de monitoramento, se existir alguma (subconsulta das linhas 1 a 10). Se não existir, o estado da entidade no dia 11 é igual ao estado atual dela, ou seja, desde o dia 11 ela não sofreu alterações na tabela monitorada (subconsulta das linhas 12 a 15).

Nesse exemplo específico, pode-se verificar pelo registro número 7 da tabela na figura 3.21 que os dados a serem retornados são exatamente os apresentados na tabela da figura 3.22, ou seja, o valor do campo “Character_level” do personagem Edgar, por exemplo, é “7”, e do campo “Strength”, 18.

Nesse modelo apresentado inicialmente, assume-se que todos os dados são inseridos na definição da tabela. Dados inseridos, nesse modelo, não tem informação temporal associada e por isso no caso de uma reconstrução da tabela em um dado ponto no tempo ou na construção da mesma como uma tabela com periodos de validade, assume-se que

todos os dados foram inseridos no mesmo instante. Por exemplo, supondo que a data de definição da tabela seja definida como 25 de Outubro de 1985, mas um registro foi inserido no dia 1° de Dezembro do mesmo ano. Uma tentativa de reconstrução da tabela utilizando os algoritmos apresentados anteriormente, no dia 28 de Novembro do mesmo ano, mostrará também o registro inserido no dia 1° de Dezembro, como se ele já existisse nesse dia, o que é incorreto.

Da mesma forma, uma tentativa de tentar traduzir a tabela de monitoramento numa tabela de tempo de validade irá gerar uma tabela no qual a informação de início de tempo de validade é a data definida como a data de criação da tabela, independente de os registros terem sido realmente inseridos nessa data ou não. Essas limitações introduzem o segundo modelo para manutenção de um registro de alterações de tempo de transação.

3.3.2 Modelo com inserções

Nesse modelo, o registro de alterações registra as inserções realizadas sobre a tabela, além de atualizações e remoções como o modelo anterior. Dessa forma, a tabela monitorada tende a ter um tamanho consideravelmente maior que no modelo anterior, sendo no mínimo igual à tabela monitorada. Além disso, esse modelo ainda apresenta algumas limitações referentes à reconstrução da tabela monitorada em situações específicas. Essas limitações serão exploradas mais tarde.

Algumas alterações são necessárias nos algoritmos do modelo anterior para lidar com o fato de que agora as inserções estão registradas na tabela de monitoramento. Para possibilitar que as inserções na tabela monitorada sejam registradas na tabela de monitoramento, basta criar uma nova *trigger* sobre a tabela monitorada para isso. Assim, qualquer inserção em qualquer ponto no tempo será registrada na tabela de monitoramento. Para manter uma uniformidade com os outros dois procedimentos já existentes, referentes às atualizações e remoções, o procedimento que registra as inserções registra apenas o identificador da entidade e a data da inserção na tabela de monitoramento, porque no caso de uma inserção não existem valores a ser perdidos. Isso poderia ser feito inserindo os valores que serão inseridos na tabela de monitoramento ao invés de vários campos nulos, mas isso gera uma confusão na tabela de monitoramento, uma vez que os valores iniciais dessa forma aparecem duplicados nessa tabela, uma vez por conta do procedimento de inserção e outra por conta da primeira atualização sobre essa entidade na tabela.

Da mesma maneira, a consulta utilizada para reconstruir a tabela monitorada em um ponto específico no tempo sofre pequenas alterações devido ao fato que agora a tabela de monitoramento contém também as inserções. A consulta anterior procura por registros que existam na tabela monitorada e não existam na tabela de monitoramento e os adiciona no resultado. Agora, a consulta adiciona os registros da tabela monitorada ou de monitoramento de acordo com as modificações que eles sofreram ao longo do tempo, baseando-se nos *timestamps* da tabela de monitoramento. A figura 3.23 exemplifica isso.

```
1  SELECT character_id, character_name, speed, strength, character_level, inteligence,
   character_class FROM c_Log_withinserts AS A WHERE NOT EXISTS ( SELECT * FROM
   c_Log_withinserts AS A2 WHERE A.character_id = A2.character_id AND DATE '2010-01-11' < A2.
   When_Changed AND A2.When_Changed < A.When_Changed) AND DATE '2010-01-11' < A.When_Changed
   AND EXISTS ( SELECT * FROM c_Log_withinserts AS A3 WHERE A.character_id = A3.character_id
   AND A3.When_Changed <= DATE '2010-01-11')
2  UNION
3  SELECT * FROM game_characters_withinserts AS P WHERE DATE '2010-01-11' > (SELECT MAX(
   When_Changed) FROM c_Log_withinserts AS A WHERE P.character_id = A.character_id)
```

Figura 3.23: Consulta para reconstruir a tabela monitorada em um ponto específico do tempo, em um modelo que armazena inserções

O exemplo mostra uma consulta para recuperar o estado da tabela monitorada no dia 11 de Janeiro de 2010. Na consulta, as tabelas receberam o sufixo "*_with_inserts*" pra identificar à qual alternativa se referem. A consulta da linha número 1 na figura 3.23 procura pelo registro imediatamente maior à data alvo. Entidades cujos valores atuais de seus atributos ainda estão na tabela monitorada na data alvo, são recuperadas pela consulta da linha número 3. Na figura 3.24 são apresentados alguns exemplos de execução dessa consulta e o estado da tabela de monitoramento.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	when_changed timestamp without time zone	character_class character varying
1	100200						2010-01-01 00:00:00	
2	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	Knight
3	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	Knight
4	100200	Edgar	5	8	2	3	2010-01-06 00:00:00	Knight
5	100200	Edgar	6	10	3	4	2010-01-07 00:00:00	Knight
6	100200	Edgar	8	13	4	5	2010-01-09 00:00:00	Knight
7	100200	Edgar	9	15	5	6	2010-01-10 00:00:00	Knight
8	100200	Edgar	11	18	6	7	2010-01-12 00:00:00	Knight
9	100200	Edgar	12	20	6	8	2010-01-13 00:00:00	Knight
10	100200	Edgar	14	23	7	9	2010-01-15 00:00:00	Knight
11	100200	Edgar	15	25	8	10	2010-01-16 00:00:00	Knight
12	100200	Edgar	17	28	9	11	2010-01-18 00:00:00	Knight
13	100200	Edgar	18	30	10	12	2010-01-19 00:00:00	Knight
14	100200	Edgar	20	33	10	13	2010-01-21 00:00:00	Knight
15	100200	Edgar	21	35	11	14	2010-01-22 00:00:00	Knight
16	100200	Edgar	23	38	12	15	2010-01-24 00:00:00	Knight
17	100200	Edgar	24	40	13	16	2010-01-25 00:00:00	Knight
18	100200	Edgar	26	43	14	17	2010-01-27 00:00:00	Knight
19	100200	Edgar	27	45	14	18	2010-01-28 00:00:00	Knight
20	100200	Edgar	29	48	15	19	2010-01-30 00:00:00	Knight
21	100200	Edgar	30	70	26	20	2010-01-31 00:00:00	Royal Guard
22	100200	Edgar	32	74	27	21	2010-02-02 00:00:00	Royal Guard
23	100200	Edgar	33	77	29	22	2010-02-03 00:00:00	Royal Guard
24	100200	Edgar	35	81	30	23	2010-02-05 00:00:00	Royal Guard
25	100200	Edgar	36	84	31	24	2010-02-06 00:00:00	Royal Guard
26	100200	Edgar	38	88	33	25	2010-02-08 00:00:00	Royal Guard
27	100200	Edgar	39	91	34	26	2010-02-09 00:00:00	Royal Guard
28	200200						2010-02-01 00:00:00	
29	200200	Sabin	1	4	1	1	2010-02-02 00:00:00	Barbarian
30	200200	Sabin	2	9	2	2	2010-02-03 00:00:00	Barbarian
31	200200	Sabin	3	14	3	2	2010-02-04 00:00:00	Barbarian
32	200200	Sabin	4	18	4	3	2010-02-05 00:00:00	Barbarian
33	200200	Sabin	5	23	5	4	2010-02-06 00:00:00	Barbarian

Tabela Monitorada em 11 de Março de 2010

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying
1	100200	Edgar	41	95	35	27	Royal Guard
2	200200	Sabin	7	32	6	7	Barbarian

Tabela Monitorada em 11 de Janeiro de 2010

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying
1	100200	Edgar	11	18	7	6	Knight

Tabela Monitorada em 04 de Fevereiro de 2010

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying
1	100200	Edgar	35	81	23	30	Royal Guard
2	200200	Sabin	4	18	3	4	Barbarian

Figura 3.24: Tabela de monitoramento e exemplos de resultados de consultas para reconstruir a tabela monitorada em um ponto específico do tempo, em um modelo que armazena inserções

A figura 3.24 apresenta o estado da tabela monitorada e três exemplos de reconstrução da tabela de monitoramento com base na mesma. A tabela de monitoramento possui dois registros com valores nulos e apenas uma data e um identificador. Esses registros são referentes às inserções realizadas sobre a tabela monitorada. Como nessa situação não existem valores a serem perdidos para serem guardados, apenas o identificador e a data da inserção são recordados. Cada exemplo da

figura 3.24 exibe uma situação distinta. No primeiro exemplo, referente ao dia 11 de Março de 2010, ambos os registros possuem os valores da tabela monitorada, pois esses são os valores válidos nessa data. No segundo exemplo, referente ao dia 11 de Janeiro de 2010, registros referentes ao personagem “Sabin” ficaram de fora do resultado da consulta porque os registros referentes a essa entidade começaram a ser inseridos no dia primeiro de Fevereiro de 2010. Por fim, no último exemplo da figura, referente ao dia 04 de Fevereiro, ambos os registros são registros da tabela de monitoramento.

Esses exemplos apresentam uma vantagem óbvia desse modelo sobre o anterior, pois é possível reconstruir com precisão a tabela monitorada no que diz respeito a inserções de dados. A tabela monitorada, nesse mesmo estado, se fosse reconstruída com os algoritmos do modelo anterior no dia 4 de Fevereiro de 2010, mostraria registros para o personagem “Sabin”, o que é sabido que é incorreto.

Da mesma forma, o algoritmo que reconstrói a tabela de monitoramento como uma tabela com períodos de validade também sofre alterações e também é mais preciso. No modelo anterior, a data de validade inicial de cada entidade era uma data definida arbitrariamente como a data de definição da tabela. Agora, é possível montar a tabela com os dados de início de validade correspondentes ao ponto no tempo em que a entidade começa a aparecer na tabela monitorada. A figura 3.25 traz a consulta que gera a tabela com períodos de validade, e o resultado da sua aplicação na tabela de monitoramento no estado em que ela é apresentada na figura 3.24.

```
1 SELECT A3.character_id, A3.character_name, A3.speed, A3.strength, A3.character_level,
2 A3.inteligence, A3.character_class, When_Changed, current_timestamp FROM c_Log_withinserts
3 AS A1, game_characters_withinserts A3 WHERE A1.character_id = A3.character_id AND NOT
  EXISTS ( SELECT * FROM c_Log_withinserts AS A2 WHERE A1.character_id = A2.character_id AND
   A1.When_Changed < A2.When_Changed)
4 UNION
5 SELECT A1.character_id, A1.character_name, A1.speed, A1.strength, A1.character_level,
6 A1.inteligence, a1.character_class, A0.When_Changed, A1.When_Changed
7 FROM c_Log_withinserts AS A0, c_Log_withinserts AS A1 WHERE A0.character_id =
8 A1.character_id AND A0.When_Changed < A1.When_Changed
9 AND NOT EXISTS (SELECT * FROM c_Log_withinserts AS M WHERE M.character_id =
10 A1.character_id AND M.When_Changed < A1.When_Changed AND M.When_Changed > A0.When_Changed)
```

	character_id integer	character_name character(100)	speed integer	strength integer	character_level integer	inteligence integer	character_class character varying	when_changed timestamp without time zone	now timestamp with time zone
1	100200	Edgar	2	3	1	1	Knight	2010-01-01 00:00:00	2010-01-03 00:00:00-0
2	100200	Edgar	3	5	2	2	Knight	2010-01-03 00:00:00	2010-01-04 00:00:00-0
3	100200	Edgar	5	8	3	2	Knight	2010-01-04 00:00:00	2010-01-06 00:00:00-0
4	100200	Edgar	6	10	4	3	Knight	2010-01-06 00:00:00	2010-01-07 00:00:00-0
5	100200	Edgar	8	13	5	4	Knight	2010-01-07 00:00:00	2010-01-09 00:00:00-0
6	100200	Edgar	9	15	6	5	Knight	2010-01-09 00:00:00	2010-01-10 00:00:00-0
7	100200	Edgar	11	18	7	6	Knight	2010-01-10 00:00:00	2010-01-12 00:00:00-0
8	100200	Edgar	12	20	8	6	Knight	2010-01-12 00:00:00	2010-01-13 00:00:00-0
9	100200	Edgar	14	23	9	7	Knight	2010-01-13 00:00:00	2010-01-15 00:00:00-0
10	100200	Edgar	15	25	10	8	Knight	2010-01-15 00:00:00	2010-01-16 00:00:00-0
11	100200	Edgar	17	28	11	9	Knight	2010-01-16 00:00:00	2010-01-18 00:00:00-0
12	100200	Edgar	18	30	12	10	Knight	2010-01-18 00:00:00	2010-01-19 00:00:00-0
13	100200	Edgar	20	33	13	10	Knight	2010-01-19 00:00:00	2010-01-21 00:00:00-0
14	100200	Edgar	21	35	14	11	Knight	2010-01-21 00:00:00	2010-01-22 00:00:00-0
15	100200	Edgar	23	38	15	12	Knight	2010-01-22 00:00:00	2010-01-24 00:00:00-0
16	100200	Edgar	24	40	16	13	Knight	2010-01-24 00:00:00	2010-01-25 00:00:00-0
17	100200	Edgar	26	43	17	14	Knight	2010-01-25 00:00:00	2010-01-27 00:00:00-0
18	100200	Edgar	27	45	18	14	Knight	2010-01-27 00:00:00	2010-01-28 00:00:00-0
19	100200	Edgar	29	48	19	15	Knight	2010-01-28 00:00:00	2010-01-30 00:00:00-0
20	100200	Edgar	30	70	20	26	Royal Guard	2010-01-30 00:00:00	2010-01-31 00:00:00-0
21	100200	Edgar	32	74	21	27	Royal Guard	2010-01-31 00:00:00	2010-02-02 00:00:00-0
22	100200	Edgar	33	77	22	29	Royal Guard	2010-02-02 00:00:00	2010-02-03 00:00:00-0
23	100200	Edgar	35	81	23	30	Royal Guard	2010-02-03 00:00:00	2010-02-05 00:00:00-0
24	100200	Edgar	36	84	24	31	Royal Guard	2010-02-05 00:00:00	2010-02-06 00:00:00-0
25	100200	Edgar	38	88	25	33	Royal Guard	2010-02-06 00:00:00	2010-02-08 00:00:00-0
26	100200	Edgar	39	91	26	34	Royal Guard	2010-02-08 00:00:00	2010-02-09 00:00:00-0
27	100200	Edgar	41	95	27	35	Royal Guard	2010-02-09 00:00:00	2012-10-03 01:03:13.4
28	200200	Sabin	1	4	1	1	Barbarian	2010-02-01 00:00:00	2010-02-02 00:00:00-0
29	200200	Sabin	2	9	2	2	Barbarian	2010-02-02 00:00:00	2010-02-03 00:00:00-0
30	200200	Sabin	3	14	2	3	Barbarian	2010-02-03 00:00:00	2010-02-04 00:00:00-0
31	200200	Sabin	4	18	3	4	Barbarian	2010-02-04 00:00:00	2010-02-05 00:00:00-0
32	200200	Sabin	5	23	4	5	Barbarian	2010-02-05 00:00:00	2010-02-06 00:00:00-0
33	200200	Sabin	6	27	5	6	Barbarian	2010-02-06 00:00:00	2010-02-07 00:00:00-0
34	200200	Sabin	7	32	7	6	Barbarian	2010-02-07 00:00:00	2012-10-03 01:03:13.4

Figura 3.25: Consulta para construção da tabela com períodos de validade

A consulta da figura 3.25 é formada por duas subconsultas. A subconsulta das linhas 1 a 3 é responsável por montar os registros responsáveis pelos valores atuais de cada

entidade. A subconsulta das linhas 5 a 11 recupera os registros passados referentes a cada entidade e monta os períodos de validade de acordo.

Em relação a consultas diretamente sobre a tabela de monitoramento, as consultas são construídas de maneira similar ao modelo anterior, com pequenas alterações, algo já demonstrado na consulta que reconstrói a tabela em um ponto específico no tempo. Um exemplo de consulta mais específica diretamente sobre a tabela de monitoramento nesse novo modelo e o resultado de sua execução são encontrados na figura 3.26 abaixo.

```
1 SELECT CHARACTER_LEVEL
2 FROM C_LOG_WITHINSERTS A WHERE
3 A.WHEN_CHANGED > TIMESTAMP '11-01-2010' AND NOT EXISTS (SELECT * FROM C_LOG_WITH_INSERTS B
   WHERE
4 B.WHEN_CHANGED > TIMESTAMP '11-01-2010'
5 AND
6 B.WHEN_CHANGED < A.WHEN_CHANGED)
7 UNION
8 SELECT CHARACTER_LEVEL FROM GAME_CHARACTERS_WITHINSERTS
9 WHERE
10 NOT EXISTS ( SELECT * FROM C_LOG_WITHINSERTS WHERE
   WHEN_CHANGED > TIMESTAMP '11-01-2010')
```

	character_level integer	character_name character(100)
1	27	Edgar
2	7	Sabin

Figura 3.26: Consulta diretamente sobre a tabela de monitoramento para obter o valor da coluna “character_level” numa data específica

Com esse modelo, é possível obter informações temporais da tabela monitorada com um nível muito maior de precisão, e a mesma possui maior flexibilidade, já que inserções podem ocorrer em qualquer momento no tempo. Entretanto, mesmo esse modelo apresenta limitações, como mencionado anteriormente. Em um cenário em que o registro referente a uma entidade seja eliminado da tabela monitorada, e posteriormente reinserido na mesma, não há uma maneira de diferenciar esses dois eventos de duas atualizações subsequentes na tabela de monitoramento. Dessa maneira, os algoritmos de reconstrução serão “enganados” e irão reconstruir a tabela monitorada com uma entidade existindo em um período em que ela deveria não existir. A figura 3.27 exemplifica melhor essa situação.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	when_changed timestamp without time zone	character_class character varying
1	200200						2010-02-01 00:00:00	
2	200200	Sabin	1	4	1	1	2010-02-02 00:00:00	Barbarian
3	200200	Sabin	2	9	2	2	2010-02-03 00:00:00	Barbarian
4	200200	Sabin	3	14	3	2	2010-02-04 00:00:00	Barbarian
5	200200	Sabin	4	18	4	3	2010-02-05 00:00:00	Barbarian
6	200200	Sabin	5	23	5	4	2010-02-06 00:00:00	Barbarian
7	200200	Sabin	6	27	6	5	2010-02-07 00:00:00	Barbarian
8	200200						2010-02-12 00:00:00	
9	200200	Sabin	7	32	7	6	2010-02-15 00:00:00	Barbarian
10	200200	Sabin	8	37	7	8	2010-02-17 00:00:00	Barbarian

Figura 3.27: Exemplo de situação em que uma remoção seguida de inserção é confundida com duas atualizações na tabela de monitoramento

A figura 3.27 mostra a situação da tabela de monitoramento em relação à entidade cujo “character_id” é 200200, no caso, o personagem de nome “Sabin”. Nessa tabela é possível verificar que existem dois registros referentes a essa entidade com valores nulos em quase todos os campos. Isso acontece porque a função de gatilho que popula a

tabela de monitoramento em inserções insere apenas o código da entidade e a data de alteração da mesma. Entretanto, essa mesma situação da tabela poderia ser reproduzida caso uma atualização fosse feita inserindo valores nulos em todos os campos a não ser no identificador, e outra atualização fosse feita restaurando valores nesses campos. Os algoritmos apresentados até agora para reconstrução da tabela irão interpretar o estado da tabela de acordo com essa segunda possibilidade, quando a interpretação correta seria que durante um espaço de tempo não existiam valores para a entidade com o identificador 200200.

Naturalmente, no exemplo acima é mais provável que o estado da tabela de monitoramento apresentado fosse resultante de uma remoção seguida por uma inserção, entretanto, algumas situações, como erros na aplicação que utiliza o banco de dados, poderiam gerar uma atualização com valores nulos. O fato é que nesse modelo, não há como distinguir remoções seguidas de inserções de atualizações subsequentes envolvendo valores nulos para todos os campos. Caso o procedimento que registra as inserções registrasse, ao invés de apenas os campos de identificação de entidade e a data da inserção, também os valores a serem inseridos, seria ainda mais complicado distinguir essas duas situações.

Para corrigir esse comportamento, é necessário expandir o modelo mais uma vez, para que ele contenha uma informação adicional na tabela de monitoramento: a operação que resultou na mudança registrada.

3.3.3 Modelo com registro de operações

Em um modelo com registro de operações, uma coluna adicional é criada na tabela de monitoramento, onde será armazenada uma informação referente a qual operação gerou aquela entrada na tabela de monitoramento. O formato dessa coluna fica a critério do projetista do banco de dados. Aqui utilizaremos uma coluna com um tipo textual que pode assumir três valores: INSERT, UPDATE e DELETE. Os procedimentos responsáveis por popular a tabela de monitoramento recebem atualizações simples para registrar valores nesse novo campo. A figura 3.28 ilustra como fica a tabela de monitoramento, agora com os novos campos.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	when_changed timestamp without time zone	operation character(10)	character_class character varying
1	100200						2010-01-01 00:00:00	INSERT	
2	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	UPDATE	Knight
3	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	UPDATE	Knight
4	100200	Edgar	5	8	2	3	2010-01-06 00:00:00	UPDATE	Knight
5	100200	Edgar	6	10	3	4	2010-01-07 00:00:00	UPDATE	Knight
6	100200	Edgar	8	13	4	5	2010-01-09 00:00:00	UPDATE	Knight
7	100200	Edgar	9	15	5	6	2010-01-10 00:00:00	UPDATE	Knight
8	100200	Edgar	11	18	6	7	2010-01-12 00:00:00	UPDATE	Knight
9	100200	Edgar	12	20	6	8	2010-01-13 00:00:00	UPDATE	Knight
10	100200	Edgar	14	23	7	9	2010-01-15 00:00:00	UPDATE	Knight
11	100200	Edgar	15	25	8	10	2010-01-16 00:00:00	UPDATE	Knight
12	100200	Edgar	17	28	9	11	2010-01-18 00:00:00	UPDATE	Knight
13	100200	Edgar	18	30	10	12	2010-01-19 00:00:00	UPDATE	Knight
14	100200	Edgar	20	33	10	13	2010-01-21 00:00:00	UPDATE	Knight
15	100200	Edgar	21	35	11	14	2010-01-22 00:00:00	UPDATE	Knight
16	100200	Edgar	23	38	12	15	2010-01-24 00:00:00	UPDATE	Knight
17	100200	Edgar	24	40	13	16	2010-01-25 00:00:00	UPDATE	Knight
18	100200	Edgar	26	43	14	17	2010-01-27 00:00:00	UPDATE	Knight
19	100200	Edgar	27	45	14	18	2010-01-28 00:00:00	UPDATE	Knight
20	100200	Edgar	29	48	15	19	2010-01-30 00:00:00	UPDATE	Knight
21	100200	Edgar	30	70	26	20	2010-01-31 00:00:00	UPDATE	Royal Guard
22	100200	Edgar	32	74	27	21	2010-02-02 00:00:00	UPDATE	Royal Guard
23	100200	Edgar	33	77	29	22	2010-02-03 00:00:00	UPDATE	Royal Guard
24	100200	Edgar	35	81	30	23	2010-02-05 00:00:00	UPDATE	Royal Guard
25	100200	Edgar	36	84	31	24	2010-02-06 00:00:00	UPDATE	Royal Guard
26	100200	Edgar	38	88	33	25	2010-02-08 00:00:00	UPDATE	Royal Guard
27	100200	Edgar	39	91	34	26	2010-02-09 00:00:00	UPDATE	Royal Guard
28	200200						2010-02-01 00:00:00	INSERT	
29	200200	Sabin	1	4	1	1	2010-02-02 00:00:00	UPDATE	Barbarian
30	200200	Sabin	2	9	2	2	2010-02-03 00:00:00	UPDATE	Barbarian
31	200200	Sabin	3	14	3	2	2010-02-04 00:00:00	UPDATE	Barbarian
32	200200	Sabin	4	18	4	3	2010-02-05 00:00:00	UPDATE	Barbarian
33	200200	Sabin	5	23	5	4	2010-02-06 00:00:00	UPDATE	Barbarian
34	200200	Sabin	6	27	6	5	2010-02-07 00:00:00	DELETE	Barbarian
35	200200						2010-02-12 00:00:00	INSERT	
36	200200	Sabin	7	32	7	6	2010-02-15 00:00:00	UPDATE	Barbarian
37	200200	Sabin	8	37	7	8	2010-02-17 00:00:00	UPDATE	Barbarian

Figura 3.28: Tabela de monitoramento com registro de operações

Naturalmente, o uso dessa nova coluna nos algoritmos de reconstrução requer que todos sejam adaptados para isso. Na figura 3.29 abaixo, é mostrada a consulta para reconstrução da tabela numa data específica, agora utilizando a nova coluna.

```
1 SELECT * FROM game_characters_recordop_beforeimages AS P WHERE DATE '2010-02-10' >
2 (SELECT MAX(When_Changed) FROM c_log_recordop_beforeimages AS A
3 WHERE P.CHARACTER_ID = A.CHARACTER_ID)
4 UNION
5 SELECT A2.CHARACTER_ID, A2.CHARACTER_NAME, A2.SPEED, A2.STRENGTH, A2.INTELIGENCE,
6 A2.CHARACTER_LEVEL, A2.CHARACTER_CLASS FROM c_log_recordop_beforeimages AS A,
  c_log_recordop_beforeimages AS A2 WHERE A.When_Changed < A2.When_Changed AND
7 A.CHARACTER_ID = A2.CHARACTER_ID AND NOT EXISTS (SELECT * FROM c_log_recordop_beforeimages
   AS A3 WHERE A.CHARACTER_ID = A3.CHARACTER_ID AND A.When_Changed < A3.When_Changed AND
8 A3.When_Changed < A2.When_Changed) AND A.When_Changed < DATE '2010-02-10' AND DATE
  '2010-02-10' < A2.When_Changed AND A.Operation <> 'DELETE'
```

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	character_class character varying
1	100200	Edgar	41	95	35	27	Royal Guard

Figura 3.29: Consulta para reconstrução da tabela monitorada, em um modelo com registro de operações

A consulta mostrada na figura 3.29 agora considera a operação especificada na tabela de monitoramento antes de incluir um registro no resultado final. Nela, as tabelas receberam o sufixo "*record_op_before_images*" para identificar à qual alternativa se referem. Os condicionais verificam se, para cada entidade, o último registro de alteração da mesma antes do ponto no tempo especificado como alvo da reconstrução é uma remoção. Se for, a entidade não existe naquele ponto no tempo e, portanto nenhum registro referente a ela é incluido no resultado final. Abaixo da consulta, está o resultado

da reconstrução da tabela no dia 10 de Fevereiro de 2010. É possível verificar que apenas um registro referente ao personagem “Edgar” é retornado, pois no período entre 7 de Fevereiro e 12 de Fevereiro de 2010, não existia nenhum registro na tabela monitorada para o personagem “Sabin”, como pode ser verificado na figura 3.28.

É possível verificar que o problema de não ser possível distinguir atualizações subsequentes de remoções seguidas de inserções não acontece. O mesmo se aplica para a consulta para geração da tabela com períodos de validade, exemplificada na figura 3.30.

```
1 select CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
  CHARACTER_CLASS, When_Changed as VALID_FROM,VALID_TO from ( SELECT A1.CHARACTER_ID,
2 A1.CHARACTER_NAME, A1.SPEED, A1.STRENGTH, A1.INTELIGENCE, A1.CHARACTER_LEVEL,
  CHARACTER_CLASS, When_Changed, current_timestamp as VALID_TO, a1.operation FROM
  c_log_recordop_beforeimages AS A1, game_characters_recordop_beforeimages WHERE
3 A1.CHARACTER_ID = game_characters_recordop_beforeimages.CHARACTER_ID AND NOT EXISTS (
  SELECT * FROM c_log_recordop_beforeimages AS A2 WHERE A1.CHARACTER_ID = A2.CHARACTER_ID
  AND A1.When_Changed < A2.When_Changed) ) as foo1 UNION select CHARACTER_ID,
  CHARACTER_NAME, SPEED,STRENGTH, INTELIGENCE, CHARACTER_LEVEL,CHARACTER_CLASS, VALID_FROM,
  end_date from (SELECT A1.CHARACTER_ID, A1.CHARACTER_NAME, A1.SPEED,A1.STRENGTH,
4 A1.INTELIGENCE, A1.CHARACTER_LEVEL, A1.CHARACTER_CLASS ,A0.When_Changed as VALID_FROM,
5 A1.When_Changed as end_date, a0.operation FROM c_log_recordop_beforeimages AS A0,
  c_log_recordop_beforeimages AS A1 WHERE A0.CHARACTER_ID = A1.CHARACTER_ID AND
6 A0.When_Changed < A1.When_Changed AND NOT EXISTS ( SELECT * FROM
  c_log_recordop_beforeimages AS M WHERE M.CHARACTER_ID = A1.CHARACTER_ID AND
7 M.When_Changed < A1.When_Changed AND M.When_Changed > A0.When_Changed)) as foo1 where
8 foo1.operation <> 'DELETE' order by CHARACTER_ID,VALID_TO
```

Figura 3.30: Consulta para tradução da tabela de monitoramento em uma tabela com períodos de validade

A consulta mostrada na figura 3.30 é igual à consulta equivalente para o modelo anterior, com apenas uma alteração específica. A tabela é montada da mesma forma que no modelo anterior, mas nesse modelo registros com o campo “*operation*” com valor “DELETE” são excluídos do resultado final, como mostrado na linha 8 da figura. Isso garante que não temos, como no modelo anterior, registros que foram excluídos fazendo parte da tabela de períodos de validade. O resultado da aplicação dessa consulta se vê na figura 3.31 abaixo.

É possível verificar que a tabela é construída corretamente, uma vez que no período entre 7 e 12 de Fevereiro não existem registros para o personagem Sabin. Esse intervalo temporal é mostrado nas linhas 33 e 34 da tabela resultante ilustrada na figura.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	character_class character varying	valid_from timestamp without time zone	valid_to timestamp with time zone
1	100200	Edgar	2	3	1	1	Knight	2010-01-01 00:00:00	2010-01-03 00:00:00-0
2	100200	Edgar	3	5	2	2	Knight	2010-01-03 00:00:00	2010-01-04 00:00:00-0
3	100200	Edgar	5	8	2	3	Knight	2010-01-04 00:00:00	2010-01-06 00:00:00-0
4	100200	Edgar	6	10	3	4	Knight	2010-01-06 00:00:00	2010-01-07 00:00:00-0
5	100200	Edgar	8	13	4	5	Knight	2010-01-07 00:00:00	2010-01-09 00:00:00-0
6	100200	Edgar	9	15	5	6	Knight	2010-01-09 00:00:00	2010-01-10 00:00:00-0
7	100200	Edgar	11	18	6	7	Knight	2010-01-10 00:00:00	2010-01-12 00:00:00-0
8	100200	Edgar	12	20	6	8	Knight	2010-01-12 00:00:00	2010-01-13 00:00:00-0
9	100200	Edgar	14	23	7	9	Knight	2010-01-13 00:00:00	2010-01-15 00:00:00-0
10	100200	Edgar	15	25	8	10	Knight	2010-01-15 00:00:00	2010-01-16 00:00:00-0
11	100200	Edgar	17	28	9	11	Knight	2010-01-16 00:00:00	2010-01-18 00:00:00-0
12	100200	Edgar	18	30	10	12	Knight	2010-01-18 00:00:00	2010-01-19 00:00:00-0
13	100200	Edgar	20	33	10	13	Knight	2010-01-19 00:00:00	2010-01-21 00:00:00-0
14	100200	Edgar	21	35	11	14	Knight	2010-01-21 00:00:00	2010-01-22 00:00:00-0
15	100200	Edgar	23	38	12	15	Knight	2010-01-22 00:00:00	2010-01-24 00:00:00-0
16	100200	Edgar	24	40	13	16	Knight	2010-01-24 00:00:00	2010-01-25 00:00:00-0
17	100200	Edgar	26	43	14	17	Knight	2010-01-25 00:00:00	2010-01-27 00:00:00-0
18	100200	Edgar	27	45	14	18	Knight	2010-01-27 00:00:00	2010-01-28 00:00:00-0
19	100200	Edgar	29	48	15	19	Knight	2010-01-28 00:00:00	2010-01-30 00:00:00-0
20	100200	Edgar	30	70	26	20	Knight	2010-01-30 00:00:00	2010-01-31 00:00:00-0
21	100200	Edgar	32	74	27	21	Royal Guard	2010-01-31 00:00:00	2010-02-02 00:00:00-0
22	100200	Edgar	33	77	29	22	Royal Guard	2010-02-02 00:00:00	2010-02-03 00:00:00-0
23	100200	Edgar	35	81	30	23	Royal Guard	2010-02-03 00:00:00	2010-02-05 00:00:00-0
24	100200	Edgar	36	84	31	24	Royal Guard	2010-02-05 00:00:00	2010-02-06 00:00:00-0
25	100200	Edgar	38	88	33	25	Royal Guard	2010-02-06 00:00:00	2010-02-08 00:00:00-0
26	100200	Edgar	39	91	34	26	Royal Guard	2010-02-08 00:00:00	2010-02-09 00:00:00-0
27	100200	Edgar	39	91	34	26	Royal Guard	2010-02-09 00:00:00	2012-10-09 04:01:51.0
28	200200	Sabin	1	4	1	1	Barbarian	2010-02-01 00:00:00	2010-02-02 00:00:00-0
29	200200	Sabin	2	9	2	2	Barbarian	2010-02-02 00:00:00	2010-02-03 00:00:00-0
30	200200	Sabin	3	14	3	2	Barbarian	2010-02-03 00:00:00	2010-02-04 00:00:00-0
31	200200	Sabin	4	18	4	3	Barbarian	2010-02-04 00:00:00	2010-02-05 00:00:00-0
32	200200	Sabin	5	23	5	4	Barbarian	2010-02-05 00:00:00	2010-02-06 00:00:00-0
33	200200	Sabin	6	27	6	5	Barbarian	2010-02-06 00:00:00	2010-02-07 00:00:00-0
34	200200	Sabin	7	32	7	6	Barbarian	2010-02-12 00:00:00	2010-02-15 00:00:00-0
35	200200	Sabin	8	37	7	8	Barbarian	2010-02-15 00:00:00	2010-02-17 00:00:00-0
36	200200	Sabin	8	37	7	8	Barbarian	2010-02-17 00:00:00	2012-10-09 04:01:51.0

Figura 3.31: Resultado da aplicação da consulta para tradução da tabela de monitoramento em uma tabela com períodos de validade

Da mesma forma que a consulta anterior, é possível verificar que a tabela é construída corretamente, ou seja, não existe tempo de validade no período em que o registro referente ao personagem Edgar foi removido e depois reinserido. O mesmo ocorre com consultas realizadas diretamente sobre a tabela de monitoramento, como visto na consulta de reconstrução da tabela monitorada em um ponto no tempo.

Parte da complexidade das consultas apresentadas até então se dá ao fato de que nas inserções da tabela de monitoramento são utilizados os valores antigos dos registros, ou seja, os valores que irão se perder após uma atualização ou remoção. Entretanto, é possível aumentar consideravelmente a simplicidade de escrita das consultas se forem utilizados os valores dos registros após a alteração que causou a inserção na tabela de monitoramento. Assim, surge uma nova alternativa de modelagem do registro de controle de alterações: utilizando imagens posteriores.

3.3.4 Modelo utilizando imagens posteriores

Nos modelos vistos até então, a tabela de monitoramento continha os valores dos registros antes deles serem alterados. Entretanto, é possível armazenar na tabela de monitoramento o estado dos campos após eles serem alterados. Assim, sempre antes de um dado ser inserido na tabela monitorada, seja via uma atualização ou uma inserção propriamente dita, ele é inserido antes na tabela de monitoramento. Essa maneira de registrar as informações na tabela de monitoramento chama-se *registro de imagens posteriores*, uma vez que é registrado o estado da tupla após a alteração.

Esse modelo exige alterações em todos os procedimentos de registro da tabela de monitoramento. O procedimento responsável por registrar as inserções agora registra os valores que irão ser inseridos na tabela. Da mesma forma, o procedimento que registra

atualizações agora insere na tabela de monitoramento os novos valores resultantes da atualização. Registros que são removidos da tabela são inseridos com valores nulos para a tabela de monitoramento, da mesma forma que as inserções nos modelos anteriores que registram inserções. A figura 3.32 mostra uma tabela de monitoramento com essas configurações.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	when_changed timestamp without time zone	operation character(10)	character_class character varying
1	100200	Edgar	2	3	1	1	2010-01-03 00:00:00	INSERT	Knight
2	100200	Edgar	3	5	2	2	2010-01-04 00:00:00	UPDATE	Knight
3	100200	Edgar	5	8	2	3	2010-01-06 00:00:00	UPDATE	Knight
4	100200	Edgar	6	10	3	4	2010-01-07 00:00:00	UPDATE	Knight
5	100200	Edgar	8	13	4	5	2010-01-09 00:00:00	UPDATE	Knight
6	100200	Edgar	9	15	5	6	2010-01-10 00:00:00	UPDATE	Knight
7	100200	Edgar	11	18	6	7	2010-01-12 00:00:00	UPDATE	Knight
8	100200	Edgar	12	20	6	8	2010-01-13 00:00:00	UPDATE	Knight
9	100200	Edgar	14	23	7	9	2010-01-15 00:00:00	UPDATE	Knight
10	100200	Edgar	15	25	8	10	2010-01-16 00:00:00	UPDATE	Knight
11	100200	Edgar	17	39	14	11	2010-01-18 00:00:00	UPDATE	Royal Guard
12	100200	Edgar	18	42	16	12	2010-01-19 00:00:00	UPDATE	Royal Guard
13	100200	Edgar	20	46	17	13	2010-01-21 00:00:00	UPDATE	Royal Guard
14	100200	Edgar	21	49	18	14	2010-01-22 00:00:00	UPDATE	Royal Guard
15	100200	Edgar	23	53	20	15	2010-01-24 00:00:00	UPDATE	Royal Guard
16	100200	Edgar	24	56	21	16	2010-01-25 00:00:00	UPDATE	Royal Guard
17	100200	Edgar	26	60	22	17	2010-01-27 00:00:00	UPDATE	Royal Guard
18	100200	Edgar	27	63	23	18	2010-01-28 00:00:00	UPDATE	Royal Guard
19	100200	Edgar	29	67	25	19	2010-01-30 00:00:00	UPDATE	Royal Guard
20	100200	Edgar	30	70	26	20	2010-01-31 00:00:00	UPDATE	Royal Guard
21	100200	Edgar	32	74	27	21	2010-02-02 00:00:00	UPDATE	Royal Guard
22	200200	Sabin	1	4	1	1	2010-01-21 00:00:00	INSERT	Barbarian
23	200200	Sabin	2	9	1	2	2010-01-22 00:00:00	UPDATE	Barbarian
24	200200	Sabin	3	14	2	3	2010-01-23 00:00:00	UPDATE	Barbarian
25	200200	Sabin	4	18	2	4	2010-01-24 00:00:00	UPDATE	Barbarian
26	200200	Sabin	5	23	3	5	2010-01-25 00:00:00	UPDATE	Barbarian
27	200200						2010-01-28 00:00:00	DELETE	
28	200200	Sabin	6	28	3	6	2010-02-01 00:00:00	INSERT	Barbarian
29	200200	Sabin	7	32	4	7	2010-02-05 00:00:00	UPDATE	Barbarian

Figura 3.32: Estado da tabela monitorada em um modelo com registro de operações e usando imagens posteriores

As consultas realizadas sobre a tabela de monitoramento sofrem alterações por conta dessa mudança. Na figura 3.33 é mostrada a consulta para obtenção do estado da tabela monitorada em um ponto específico no tempo, agora utilizando imagens posteriores.

```
1 SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
  CHARACTER_CLASS FROM c_log_recordop_afterimages AS A
2 WHERE A.When_Changed =(SELECT MAX(A2.When_Changed) FROM
3 c_log_recordop_afterimages AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A2.
  When_Changed < DATE '2010-01-26') AND A.Operation <> 'DELETE'
```

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	character_class character varying
1	100200	Edgar	24	56	21	16	Royal Guard
2	200200	Sabin	5	23	3	5	Barbarian

Figura 3.33: Consulta para reconstrução da tabela monitorada em um ponto específico no tempo

Na consulta da figura 3.33, a tabela monitorada, com o estado idêntico ao mostrado na figura 3.32, é reconstruída no dia 26 de Janeiro de 2010. Nela, as tabelas receberam o sufixo "*record_op_after_images*" para identificar à qual alternativa se referem.

Pela tabela, é possível ver que essa simples consulta reconstrói com correção o estado da tabela monitorada. Comparando essa consulta com a consulta do modelo anterior usada para o mesmo fim, é possível ver que essa possui um grau de simplicidade maior, uma vez que nesse caso todos os valores estão disponíveis na tabela de monitoramento.

Essa constatação também se aplica em consultas para reconstruir a tabela monitorada com períodos de validade. A figura 3.34 apresenta uma consulta desse tipo e o resultado de sua execução.

```
1 SELECT A.CHARACTER_ID, A.CHARACTER_NAME, A.SPEED, A.STRENGTH, A.INTELIGENCE, A.
2 CHARACTER_LEVEL, A.CHARACTER_CLASS, A.When_Changed AS VALID_FROM, A2.When_Changed AS
  VALID_TO
  FROM c_log_recordop_afterimages AS A, c_log_recordop_afterimages AS A2 WHERE A.
3 CHARACTER_ID = A2.CHARACTER_ID AND A.When_Changed < A2.When_Changed AND A.Operation <>
  'DELETE'
  AND NOT EXISTS ( SELECT * FROM c_log_recordop_afterimages AS A3 WHERE A.CHARACTER_ID = A3.
4 CHARACTER_ID AND A.When_Changed < A3.When_Changed AND A3.When_Changed < A2.When_Changed)
  UNION
5 SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
6 CHARACTER_CLASS,  When_Changed, CURRENT_DATE FROM c_log_recordop_afterimages
  AS A WHERE A.Operation <> 'DELETE'
7 AND NOT EXISTS ( SELECT * FROM c_log_recordop_afterimages AS A3 WHERE A.CHARACTER_ID = A3.
8 CHARACTER_ID AND A.When_Changed < A3.When_Changed )
```

Figura 3.34: Consulta para reconstrução da tabela monitorada em uma tabela com períodos de validade

A figura 3.34 ilustra a consulta para reconstruir a tabela monitorada como uma tabela com períodos de validade. É possível verificar na mesma a montagem de cada grupo de registro, agrupados com uma cláusula UNION na linha 4. A primeira subconsulta monta os registros intermediários ao longo do tempo, enquanto que a segunda subconsulta recupera os registros com os valores atuais dos atributos. Utilizando imagens posteriores não é necessário buscar valores na tabela monitorada porque todos os valores estão disponíveis na tabela de monitoramento. A figura 3.35 ilustra o resultado da aplicação dessa consulta sobre uma tabela de monitoramento com o estado ilustrado na figura 3.32.

	character_id integer	character_name character(100)	speed integer	strength integer	inteligence integer	character_level integer	character_class character varying	valid_from timestamp without time zone	valid_to timestamp without time zone
1	100200	Edgar	2	3	1	1	Knight	2010-01-03 00:00:00	2010-01-04 00:00:00
2	100200	Edgar	3	5	2	2	Knight	2010-01-04 00:00:00	2010-01-06 00:00:00
3	100200	Edgar	5	8	2	3	Knight	2010-01-06 00:00:00	2010-01-07 00:00:00
4	100200	Edgar	6	10	3	4	Knight	2010-01-07 00:00:00	2010-01-09 00:00:00
5	100200	Edgar	8	13	4	5	Knight	2010-01-09 00:00:00	2010-01-10 00:00:00
6	100200	Edgar	9	15	5	6	Knight	2010-01-10 00:00:00	2010-01-12 00:00:00
7	100200	Edgar	11	18	6	7	Knight	2010-01-12 00:00:00	2010-01-13 00:00:00
8	100200	Edgar	12	20	6	8	Knight	2010-01-13 00:00:00	2010-01-15 00:00:00
9	100200	Edgar	14	23	7	9	Knight	2010-01-15 00:00:00	2010-01-16 00:00:00
10	100200	Edgar	15	25	8	10	Knight	2010-01-16 00:00:00	2010-01-18 00:00:00
11	100200	Edgar	17	39	14	11	Royal Guard	2010-01-18 00:00:00	2010-01-19 00:00:00
12	100200	Edgar	18	42	16	12	Royal Guard	2010-01-19 00:00:00	2010-01-21 00:00:00
13	100200	Edgar	20	46	17	13	Royal Guard	2010-01-21 00:00:00	2010-01-22 00:00:00
14	100200	Edgar	21	49	18	14	Royal Guard	2010-01-22 00:00:00	2010-01-24 00:00:00
15	100200	Edgar	23	53	20	15	Royal Guard	2010-01-24 00:00:00	2010-01-25 00:00:00
16	100200	Edgar	24	56	21	16	Royal Guard	2010-01-25 00:00:00	2010-01-27 00:00:00
17	100200	Edgar	26	60	22	17	Royal Guard	2010-01-27 00:00:00	2010-01-28 00:00:00
18	100200	Edgar	27	63	23	18	Royal Guard	2010-01-28 00:00:00	2010-01-30 00:00:00
19	100200	Edgar	29	67	25	19	Royal Guard	2010-01-30 00:00:00	2010-01-31 00:00:00
20	100200	Edgar	30	70	26	20	Royal Guard	2010-01-31 00:00:00	2010-02-02 00:00:00
21	100200	Edgar	32	74	27	21	Royal Guard	2010-02-02 00:00:00	2012-10-11 00:00:00
22	200200	Sabin	1	4	1	1	Barbarian	2010-01-21 00:00:00	2010-01-22 00:00:00
23	200200	Sabin	2	9	1	2	Barbarian	2010-01-22 00:00:00	2010-01-23 00:00:00
24	200200	Sabin	3	14	2	3	Barbarian	2010-01-23 00:00:00	2010-01-24 00:00:00
25	200200	Sabin	4	18	2	4	Barbarian	2010-01-24 00:00:00	2010-01-25 00:00:00
26	200200	Sabin	5	23	3	5	Barbarian	2010-01-25 00:00:00	2010-01-28 00:00:00
27	200200	Sabin	6	28	3	6	Barbarian	2010-02-01 00:00:00	2010-02-05 00:00:00
28	200200	Sabin	7	32	4	7	Barbarian	2010-02-05 00:00:00	2012-10-11 00:00:00

Figura 3.35: Resultado da aplicação de consulta para reconstrução da tabela monitorada em uma tabela com períodos de validade

Mais uma vez se verifica que a consulta nesse modelo utilizando imagens posteriores é mais simples de ser escrita quando comparada ao modelo anterior, pelo mesmo motivo, não necessita cruzar as tabelas monitorada e de monitoramento.

Naturalmente, isso é um indício de que esse modelo é mais adequado mas apenas essas informações não são suficientes para eleger um modelo “melhor” ou “pior”. Para atestar isso, é necessário verificar, por exemplo, o desempenho das consultas em cada modelo e qual o volume de dados gerado em cada um. Um estudo sobre isso é apresentado no próximo capítulo, onde todos os modelos de representação de dados temporais apresentados nesse capítulo são analisados de um ponto de vista de desempenho e usabilidade nos mais diversos contextos.

3.4 Outras alternativas propostas na literatura

Os modelos apresentados até então nesse capítulo são alguns dos utilizados nos projetos de bases de dados temporais usando bancos de dados puramente relacionais. Entretanto, existem outras alternativas na literatura para lidar com dados em bases temporais.

Muitas abordagens (SNODGRASS, 1987; MKAOUAR, 2011) se baseiam em extensões para a linguagem SQL padrão, para cobrir as lacunas existentes na mesma em relação a dados temporais, especialmente a ausência de períodos, do conceito de agora e do conceito de infinito.

Existem também referências para frameworks de projeto e desenvolvimento de base de dados temporais. Bergamaschi (1998), por exemplo, propõe uma maneira automatizada de definir entidades temporais, e o framework, entre outras coisas, constrói automaticamente os procedimentos referentes a restrições de integridade citados na seção 3.2.2, enquanto Moro (2002) apresenta um modelo temporal de versões onde o usuário define classes temporais e atemporais utilizando uma interface própria, e essas definições são traduzidas em SQL convencional por meio de uma camada entre essa interface e o SGBD.

Outras abordagens encontradas na literatura são tradutores e extensões em XML, onde o projetista especifica atributos e entidades temporais através de arquivos XML e esses são posteriormente traduzidos para construtores SQL padrão. Manica (2009) descreve uma ferramenta capaz de realizar consultas em TSQL2 (SNODGRASS, 1995) sobre SGBDs relacionais. Wang (2008) define um sistema onde entidades temporais e construções de histórico da base de dados são definidas em XML e consultadas através de XQuery. Já Noh (2008-a) define uma arquitetura onde também são usados arquivos XML para definir as entidades da base de dados, mas ao invés de usar um banco de dados relacional como base, é utilizado um modelo de armazenamento próprio. Por fim, JOHNSTON (2010) define um framework intermediário a ser aplicado sobre todas as tabelas de uma base de dados, onde todas são tratadas como tabelas bitemporais, automaticamente possuindo suporte à tempo de transação e tempo de validade, possibilitando auditorias e consultas temporais sobre as mesmas. Esse framework é chamado pelo autor de *Asserted Versioning*.

Ainda existem modelos de dados temporais não cobertos nesse trabalho. É possível ter tabelas com mais de um campo temporal representando cada entidade, no caso de haver registro e distinção entre tempo de validade e tempo de transação na mesma tabela (SNODGRASS, 1987).

Nenhuma das alternativas apresentadas nessa seção supõe o uso apenas de um SGBD relacional tradicional. Por isso essas alternativas não são estudadas mais a fundo

nesse trabalho, uma vez que ele se propõe ao estudo de alternativas que utilizem apenas o banco de dados relacional “puro”.

Um trabalho similar ao desenvolvido aqui, mas sobre bases de dados objeto relacionais, é em encontrado em (ATAY, 2010). Nesse trabalho, é realizado um estudo comparativo de alternativas utilizando *atribute-timestamping* e *tuple-timestamping*, conceitos que também são abordados no estudo desenvolvido aqui. Entretanto, a comparação feita aqui é entre diferentes maneiras de aplicar *timestamping* sobre atributos ou tuplas, enquanto que no trabalho citado essas duas abordagens são comparadas entre sí.

O projetista e o desenvolvedor de aplicações de bases de dados enfrentam recorrentemente os problemas e escolhas apresentadas nesse capítulo, o que leva a necessidade de fazer uma escolha correta dependendo do seu contexto e das vantagens e desvantagens que cada modelo oferece. Um estudo aprofundado sobre cada modelo é apresentado no capítulo seguinte.

4 EXPERIMENTOS SOBRE AS ALTERNATIVAS APRESENTADAS

No capítulo anterior, foram apresentadas alternativas para representar dados temporais. Muitas vezes, ao projetar uma base de dados, o projetista tem várias das alternativas apresentadas como opção, mas muitas vezes a pergunta “qual a melhor alternativa para o nosso contexto” não é simples de ser respondida. Instintivamente o projetista pode até ter uma ideia, mas uma decisão dessa importância necessita ser a mais precisa possível, porque o banco de dados resultante sofrerá as consequências dessa decisão durante toda a sua vida útil.

Com o objetivo de fazer uma comparação objetiva entre as alternativas apresentadas no capítulo anterior, foram realizados experimentos sobre cada uma delas, comparando-as utilizando diversos critérios. Para tanto, foi necessário a montagem de um grande framework experimental, além de uma configuração de sistema que possibilitasse testar todas as alternativas em igualdade de condições, e facilitasse a análise de resultados.

4.1 Configurações

Os experimentos foram rodados sobre um sistema com as seguintes características:

- Processador Intel® Core™2 Duo CPU T5450 1.66Ghz
- 2GB de memória RAM
- Sistema Operacional Windows Vista 6.0 (Build 6002: Service Pack 2)

Nesse sistema tradicional e pouco potente, as características de cada modelo pesquisado, em termos de eficiência, ficam ainda mais evidentes do que ficariam em um modelo mais poderoso, onde um grande poder computacional poderia amenizar eventuais problemas de performance apresentados por alguma das alternativas estudadas.

O SGBD utilizado para rodar os experimentos foi o PostgreSQL 9.0. Ele foi escolhido porque suporta grande parte das estruturas temporais especificadas na linguagem SQL, além de ser um sistema robusto, permitindo a execução e avaliação eficaz dos experimentos propostos nesse trabalho.

Todos os experimentos foram rodados em condições idênticas, para garantir que não haveriam influencias externas, como processos concorrentes rodando no mesmo sistema, que pudessem influenciar os resultados obtidos.

4.2 Metodologia

Para garantir que os experimentos rodassem sobre cenários próximos da realidade, foi definida uma metodologia para a aplicação dos testes. Na avaliação dos resultados são utilizados além de critérios objetivos, como desempenho de execução, critérios subjetivos como simplicidade de construção de consultas para a base de dados alvo.

Todas as bases de dados sobre as quais foram rodados experimentos foram populadas massivamente com dados artificiais. Para tanto, foram escritos programas parametrizáveis com as quantidades desejadas de dados em cada uma das tabelas alvo. Esses programas permitiram que os experimentos fossem refinados após múltiplas execuções, para que se tivesse um cenário onde fosse possível fazer uma análise de cada alternativa definida.

Após isso, são executados conjuntos de consultas sobre a base de dados alvo, e o tempo de execução de cada consulta é armazenado para posterior avaliação. Para garantir uma maior precisão nos resultados, esse ciclo que começa com a população da base de dados, continua com a execução das consultas e termina com a coleta de resultados e limpeza da base, é executado seis vezes, e uma média dos resultados é calculada para obter o resultado final. Esse número de execuções foi definido a partir de experimentaçao, pois se verificou que esse número é suficiente para eliminar eventuais discrepâncias na execução dos testes. Todo o ciclo de coleta de dados é executado automaticamente.

Uma vez que todas as execuções foram concluídas, as avaliações sobre os resultados foram feitas, utilizando os critérios pré-estabelecidos. Os critérios utilizados para avaliar cada alternativa são detalhados na seção seguinte.

4.3 Critérios

Para avaliar todas as alternativas estudadas nesse trabalho, foram utilizados critérios julgados relevantes ao se avaliar uma aplicação de base de dados, com base em referências encontradas na literatura (ATAY, 2010; SCHIEFER, 1999, GRAEFE, 1991). Esses critérios são divididos em critérios objetivos, para os quais uma análise numérica é feita, e critérios subjetivos, que são analisados para obtenção de um parecer sobre o comportamento de cada alternativa em cima de cada critério.

Os critérios objetivos usados são:

- Volume de dados
- Replicação de dados
- Desempenho geral de consultas

Já os critérios subjetivos são:

- Simplicidade para escrita de novas consultas
- Necessidade de verificações de integridade externas

O volume de dados gerado pelo uso de cada alternativa é avaliado comparando a diferença de volume de dados entre as alternativas quando elas representam dados similares. Por exemplo, supondo que numa comparação entre modelos para representação de entidades temporais a base de dados seja populada com dados referentes a cem entidades, podem haver diferenças no volume de dados das tabelas de cada modelo, mesmo que os dados de cada modelo se refiram às mesmas entidades. Existem várias razões para isso, diretamente relacionadas com as características de cada modelo, e é exatamente isso que é analisado em detalhe em cada experimento.

Da mesma maneira, o nível de replicação de dados também é diretamente afetado pelas características de cada modelo. Alguns modelos fazem com que dados sejam

replicados mais vezes, enquanto outros são mais econômicos nesse sentido. Esse é um critério ligado ao critério de volume de dados, uma vez que ele gera um aumento nesse volume, mas não é o único responsável por diferença de volume entre as alternativas. Como usualmente sempre se tenta diminuir a replicação de dados, esse critério é analisado separadamente do volume.

Já o desempenho geral de consultas é um critério geral, mas especializado de acordo com os diferentes tipos de consultas aplicáveis a cada alternativa de representação de dados temporais. Por exemplo, em modelos que podem enfrentar o problema de tabelas temporais referenciadas, existem consultas que testam o desempenho de cada modelo em cima dos tipos de consultas que são afetadas por esse tipo de restrição. O resultado de cada consulta é consolidado num resultado geral, que é utilizado para comparar cada modelo em termos de eficiência.

Além dos critérios objetivos detalhados anteriormente, dois critérios subjetivos também são empregados para analisar cada alternativa. O primeiro deles é a simplicidade para escrita de consultas para um determinado modelo. Aplicações sobre bases de dados são dinâmicas e frequentemente é necessário fazer alterações em consultas já existentes ou criar novas consultas. A facilidade com que essas novas consultas são criadas, ou consultas existentes alteradas, é muito importante durante o ciclo de vida do sistema. É mais difícil entender e escrever consultas excessivamente complexas, o que frequentemente leva a erros.

Outro critério subjetivo utilizado para avaliar cada alternativa estudada é a necessidade de verificações e procedimentos externos aos oferecidos pelo SGBD para manter a integridade da base de dados. Naturalmente que a existência desse tipo de verificação sobre as tabelas da base de dados tende a afetar o desempenho de consultas sobre as tabelas, mas essa não é a única consequência. Sempre que existe a necessidade de escrever código para fazer alguma verificação de integridade, existe a necessidade de que esse código sofra manutenção, sendo esse mais um ponto propício a erros. Quanto mais complicado de se escrever esse código, maior a possibilidade de que ele seja escrito erroneamente. No fim, esse tipo de código acaba sendo mais uma preocupação para o desenvolvedor que trabalha com a base de dados.

4.4 Experimentos

Para analisar em detalhe e possibilitar uma comparação fundamentada sobre as alternativas apresentadas no capítulo anterior, foram realizados experimentos sobre todas as alternativas, seguindo a metodologia explicada nas sessões anteriores desse capítulo.

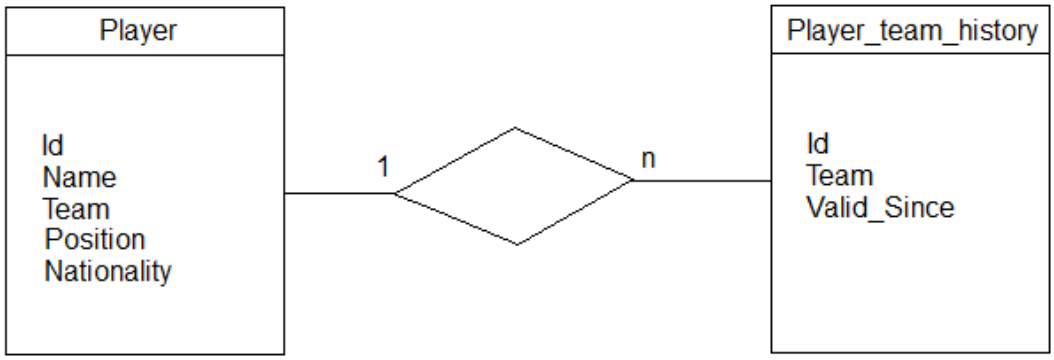
4.4.1 Alternativas para representação de atributos temporais

No capítulo anterior, foram discutidas duas abordagens para representar dados de atributos temporais: utilizando instantes ou períodos. Além disso, foi mostrada a possibilidade de separar os dados do atributo temporal de outros dados não temporais da entidade. Com base nisso, foram formulados quatro conjuntos diferentes de tabelas, cada um deles aplicando uma combinação de uma das alternativas de representação de dados com o uso ou não de separação de dados temporais de dados atemporais.

Cada um dos conjuntos de tabelas foi rotulado com um nome no formato “SolutionX”, onde “X” é um número de 1 a 4. Essas quatro alternativas foram

comparadas de acordo com os critérios apresentados na seção anterior e os resultados são discutidos nas subseções seguintes. As soluções representam os dados de um jogador de futebol, com o seu time variando no tempo, assim como nos exemplos do capítulo anterior.

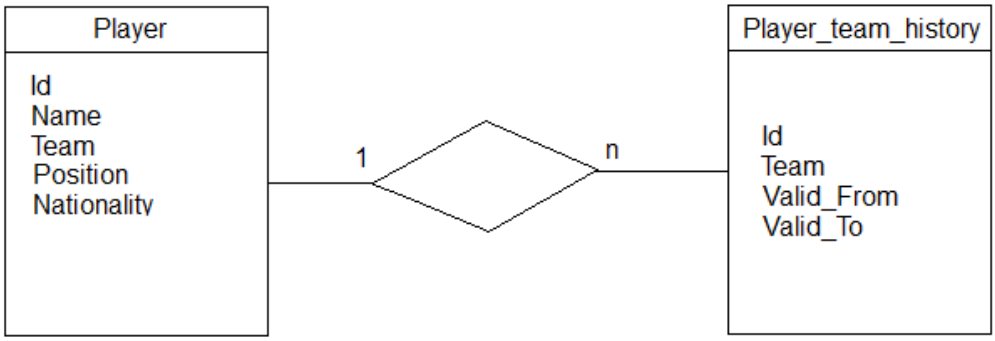
A primeira alternativa analisada, rotulada “Solution1”, utiliza apenas um campo temporal para representar a validade do valor do atributo “*team*”, e separa dados temporais de dados atemporais. A figura 4.1 abaixo apresenta o modelo ER correspondente a essa alternativa.



Player(**Id** ,Name, Team, Position, Nationality)
Player_team_history (**Id** , Team, **Valid_Since**)

Figura 4.1: Modelo ER e representação relacional da alternativa #1

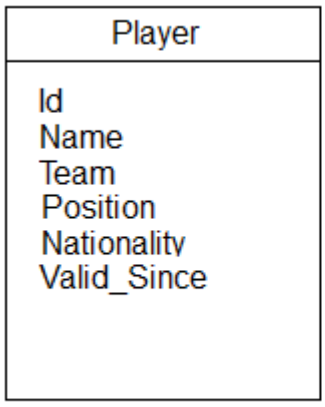
Já a segunda alternativa, rotulada “Solution2”, apresenta estrutura similar à apresentada na alternativa anterior, mas com dois campos para indicar a validade da informação de time de cada jogador. Ela também aplica separação entre dados temporais e atemporais. O ER correspondente e a descrição das tabelas no modelo relacional aparecem na figura 4.2 seguinte.



Player(**Id** ,Name, Team, Position, Nationality)
Player_team_history(**Id** , Team, **Valid_From, Valid_To**)

Figura 4.2: Modelo ER e representação relacional da alternativa #2

Ambas as alternativas que utilizam tabelas separadas utilizam a otimização mencionada no capítulo anterior referente a armazenar o valor corrente do atributo temporal na tabela com os atributos atemporais. Já as alternativas rotuladas “Solution3” e “Solution4” não separam dados temporais de atemporais. “Solution3” utiliza apenas um campo para representar a validade do atributo temporal, como mostrado na figura 4.3.



Player(**Id** ,Name, Team, Position, Nationality, **Valid_Since**)

Figura 4.3: Modelo ER e representação relacional da alternativa #3

Já a alternativa rotulada “Solution4” utiliza dois campos para representar a validade dos dados temporais. O ER correspondente aparece na figura 4.4 abaixo.

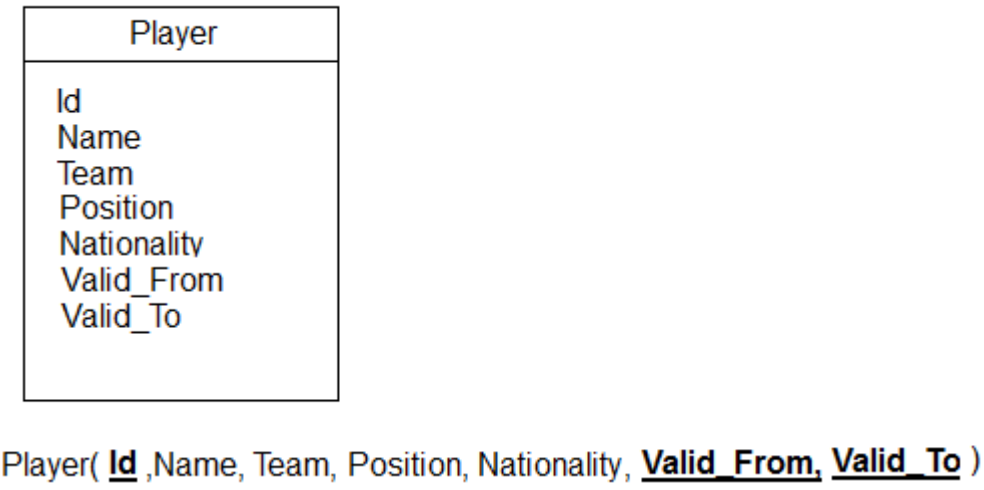


Figura 4.4: Modelo ER e representação relacional da alternativa #4

4.4.1.1 Preparação

Uma vez definidas as alternativas, elas foram implementadas e populadas para serem analisadas por meio de experimentos. Primeiramente, cada conjunto de tabelas foi populado com dados representando os elencos de aproximadamente dois mil e trezentos times, cada um dos times com um número de jogadores variando entre trinta e trinta e oito. Isso representa um total aproximado de sessenta e nove mil entidades representadas nas tabelas de cada solução, todas convivendo na mesma base de dados.

Para cada jogador, foi definido que seriam criados seis registros de times anteriores, referentes aos times pelos quais o jogador havia passado antes de chegar ao seu time atual. O período que cada jogador esteve em cada time foi gerado aleatoriamente, variando entre um a quatro anos. Com essas configurações, o número de registros em cada uma das soluções ultrapassa os quatrocentos mil em qualquer das alternativas de representação. No sistema sobre o qual os experimentos foram rodados, esse número é mais do que suficiente para que a quantidade de dados nas tabelas de cada alternativa realce as virtudes e desvantagens de cada uma delas.

É após esse momento em que todas as tabelas envolvidas foram populadas, que foram aplicados os critérios de avaliação de volume de dados e de replicação de dados. Uma comparação foi feita entre cada alternativa de representação para verificar o quanto cada alternativa ocupa em termos de volume de dados para representar os dados referentes às mesmas entidades. Da mesma forma, foi verificado o quanto havia em termos de dados replicados nas tabelas de cada alternativa, e os resultados foram computados para posterior análise.

4.4.1.2 Análise sobre o Volume de Dados

Uma vez que as tabelas foram populadas, elas foram analisadas em função do volume de dados de cada alternativa. A Tabela 4.1 abaixo mostra quanto espaço em disco as tabelas referentes a cada alternativa ocuparam.

Tabela 4.1: Comparação entre o tamanho de cada tabela

	Tabelas Separadas		Mesma Tabela	
	Solution1	Solution2	Solution3	Solution4
Tabela principal	18 MB	18 MB	149 MB	206 MB
Tabela de histórico	56 MB	91 MB	N/A	N/A
Tamanho total	74 MB	109 MB	149 MB	206 MB

A Tabela 4.1 é organizada da seguinte forma: a primeira coluna apresenta uma distinção entre a tabela principal, tabela de histórico e o tamanho total dos dados. As duas colunas seguintes apresentam os dados referentes a alternativas que de fato utilizam uma tabela separada para os dados históricos, e a as duas restantes para as alternativas que possuem apenas uma tabela.

É possível notar, pelos dados contidos na Tabela 4.1, que as soluções que não separam dados temporais de atemporais possuem um aumento expressivo de tamanho. Isso se dá especialmente em função da redundância de dados atemporais nessas tabelas. Levando em consideração que são três atributos atemporais, é plausível atribuir, de acordo com os números obtidos no experimento, algo em torno de 30% a 33% para cada um no aumento de tamanho da tabela quando campos atemporais e temporais não são separados. Claro que o tipo de dado armazenado pode influenciar no quanto de aumento de volume um campo atemporal provoca. No caso dos experimentos, todos os campos são textuais, portanto pode-se afirmar que eles têm peso igual no aumento de volume registrado.

É interessante que o projetista tenha em mente esses números ao tomar uma decisão, já que em um contexto onde as entidades possuem muitos campos atemporais pode ser prudente separá-los dos campos temporais, mesmo com uma possível perda de performance ocasionada pelo fato de que ter uma tabela separada envolve uma junção entre duas tabelas nas consultas. Entretanto, mesmo a performance pode ser beneficiada por uma separação entre campos temporais e atemporais, uma vez que um aumento expressivo no volume de dados pode fazer com o que o banco tenha mais dificuldade em lidar com os dados armazenados, causando um decréscimo de performance mais sensível que o notado pela junção entre as duas tabelas. Isso é analisado na seção seguinte.

Um aumento considerável é notado também em função do campo temporal extra nas alternativas 2 e 4, em torno de 25%. Um campo temporal tende a ocupar espaço em disco menor que um campo textual, por isso a influência desse tipo de campo no aumento de volume da base em tese não deveria ser relevante. Entretanto, campos temporais em geral são componentes de uma chave primária, como no caso das alternativas estudadas. Um campo temporal a mais gera um novo índice que por consequência acaba gerando um acréscimo no volume de dados. Isso explica o número expressivo em termos de aumento de volume causado por esse campo. Como soluções baseadas em períodos não podem depender apenas de chaves primárias para assegurar a integridade dos dados, ou seja, dependem de verificações externas, o projetista pode optar por abrir mão de usar chaves primárias nessas tabelas, economizando espaço em disco, mas possivelmente perdendo em performance.

A alternativa mais econômica em termos de espaço em disco (Solution1) apresenta um volume de dados 60% menor que a alternativa mais custosa (Solution4). Isso demonstra a grande influência que a modelagem dos dados tem sobre o volume das tabelas que lidam com atributos temporais. De posse desses dados concretos sobre esse impacto, o projetista pode fazer uma escolha orientada de qual modelagem utilizar, dependendo do seu contexto.

4.4.1.3Análise sobre a necessidade de verificações de integridade externas

Além dos comandos referentes à criação das tabelas, e dos comandos para extrair ou alterar os dados presentes nas mesmas, é necessário que a solução adotada pelo projetista garanta a integridade dos dados presentes nela. Dependendo da alternativa de representação escolhida, isso irá requerer que um procedimento externo garanta essa integridade ou não.

No capítulo anterior, quando foram definidas as diferenças entre a representação da validade do atributo temporal utilizando apenas um instante inicial ou um período, foi mostrado que soluções que utilizam apenas instantes podem utilizar construtores oferecidos pelo SGBD, como chaves primárias, para garantir essa integridade, pois por definição dois registros referentes a mesma entidade e com a mesma data de início não podem existir.

Já com soluções que utilizam períodos, o problema é mais complexo. Deve-se garantir que os períodos de validade não se interseccionem em nenhum instante no tempo, já que para instantes onde há intersecção não será possível saber qual é o dado válido. Nesse caso, se utilizam verificações externas quando inserções ou atualizações são feitas nas tabelas que contém os dados temporais para garantir que os dados sendo inseridos ou alterados não violam as restrições de integridade. Esses procedimentos podem ser executados pelas aplicações externas que consomem dados das tabelas das soluções ou através de *triggers* sobre as tabelas na própria base de dados. Deixar a verificação a cargo das aplicações consumidoras é extremamente perigoso, uma vez que toda e qualquer aplicação acessando e alterando a base de dados teria que ter as verificações de integridade embutidas, o que em ambientes reais é algo difícil de garantir. Por isso, na grande maioria dos casos, os procedimentos que garantem a integridade dos dados são implementados usando *triggers* ligadas às tabelas com dados temporais.

No caso das alternativas estudadas, as alternativas que necessitam desses procedimentos são as alternativas 2 e 4, que utilizam períodos para representar a validade dos dados temporais. Qualquer alteração sobre as tabelas dessas alternativas que possuem dados temporais necessita ser checada para garantir que a alteração não provoque quebra na integridade dos dados no que diz respeito aos intervalos de validade. Na figura 4.5 é possível verificar como fica a defínção das *triggers* associadas com esses procedimentos.

```
1  CREATE TRIGGER <TRIGGER_NAME>
2    BEFORE INSERT
3    ON solution4
4    FOR EACH ROW
5    EXECUTE PROCEDURE <PROCEDURE_NAME>;
6
7  CREATE TRIGGER <TRIGGER_NAME>
8    BEFORE UPDATE
9    ON solution4
10   FOR EACH ROW
11   EXECUTE PROCEDURE <PROCEDURE_NAME>;
```

Figura 4.5: Definição das *triggers* que disparam procedimentos de verificação de integridade em tabelas que utilizam períodos para representação da validade de dados temporais.

As definições da figura 4.5 são bastante simples. Entretanto é importante notar algumas características de cada *trigger*. Ambas as *triggers* devem ser definidas como "*BEFORE*" para que a verificação seja executada sobre as tabelas antes delas sofrerem alterações (linhas 2 e 8 da figura) e "*FOR EACH ROW*" para garantir que todas as linhas alteradas sejam verificadas. Esse segundo passo é essencial no caso de uma atualização.

Já na figura 4.6 abaixo é possível verificar como pode ser implementado um procedimento de verificação de integridade de acordo com as regras mencionadas anteriormente. No caso dessa figura, o procedimento é referente a uma inserção.

```
1  DECLARE
2      R RECORD;
3      BEGIN
4      FOR R IN SELECT * FROM <HISTORY_TABLE> WHERE ID = NEW.ID AND NEW.VALID_FROM IS NOT
       NULL LOOP
5
6              IF (NEW.VALID_FROM,NEW.VALID_TO) OVERLAPS (R.VALID_FROM,R.VALID_TO) THEN
7                  RAISE EXCEPTION 'INTERVAL CONFLICT! ID= %, VALID_FROM=%, VALID_TO=%', NEW.
                   ID, NEW.VALID_FROM, NEW.VALID_TO;
8              END IF;
9
10         END LOOP;
11         RETURN NEW;
12     END;
```

Figura 4.6: Código PgPL/SQL para garantir integridade dos dados, em inserções, em tabelas com dados temporais em soluções que utilizam períodos

É possível verificar que o código apresentado na figura 4.6 é bastante simples em termos de lógica. Ele seleciona todos os registros referentes à mesma entidade para à qual está sendo inserido um novo registro na tabela, identificada na figura pelo literal "*<HISTORY_TABLE>*", e verifica se a validade do registro novo não intersecciona a validade de um registro existente. A verificação de intersecção é bastante fácil de ser implementada, já que todos os registros que estão sendo testados possuem um período de validade associado. No código da figura acima, foi utilizado o operador *overlaps* para isso, mas mesmo em SGBDs que não suportam esse operador ainda seria bastante simples de se implementar essa verificação checando os limites dos períodos através de operadores básicos como ">=" (maior ou igual). No caso de uma atualização, cada um dos registros alterados passa por essa verificação. Já no caso de uma inserção, apenas o registro que está sendo inserido é verificado. A figura 4.7 abaixo mostra como fica o procedimento para verificar atualizações.

```
1  DECLARE
2      R RECORD;
3      BEGIN
4      FOR R IN SELECT * FROM <HISTORY_TABLE> WHERE ID = NEW.ID AND NEW.VALID_FROM IS NOT
       NULL LOOP
5          IF (NOT (OLD.VALID_FROM = R.VALID_FROM)) THEN
6              IF (NEW.VALID_FROM,NEW.VALID_TO) OVERLAPS (R.VALID_FROM,R.VALID_TO) THEN
7                  RAISE EXCEPTION 'INTERVAL CONFLICT! ID= %, VALID_FROM=%, VALID_TO=%', NEW.
                   ID, NEW.VALID_FROM, NEW.VALID_TO;
8              END IF;
9          END IF;
10         END LOOP;
11         RETURN NEW;
12     END;
```

Figura 4.7: Código PgPL/SQL para garantir integridade dos dados, em atualizações, em tabelas com dados temporais em soluções que utilizam períodos

O procedimento acima é muito similar ao procedimento utilizado para verificar inserções, com apenas a mudança que o registro que está sendo alterado pela atualização não é testado, já que não faz sentido testar o registro novo gerado pela atualização com o mesmo registro, só que com os valores antigos. Isso é feito através do teste da linha 5, que verifica se o registro "*old*", que contém os valores antigos do registro sendo testado, não é o mesmo que que está prestes a ser verificado e está na variável R.

Apesar da lógica extremamente simples, um procedimento desse tipo pode envolver a checagem de muitos registros, podendo causar impactos de performance, especialmente em casos de atualizações, onde muitos registros são envolvidos. Na subseção seguinte é analisada a performance de cada alternativa em face a diferentes tipos de consultas e comandos. O possível impacto gerado pela existência desse tipo de procedimento também é levado em consideração.

4.4.1.4 Execução e Análise de Simplicidade de Consultas

Uma vez que as tabelas referentes a cada solução foram populadas, foram rodadas consultas sobre cada conjunto de tabelas referente a cada alternativa. Cada uma das consultas formuladas tem por objetivo testar o desempenho de cada uma das alternativas em um tipo específico de operação. As consultas são adaptadas para as características de cada uma das alternativas, mas semanticamente as consultas para cada tipo de operação são sempre equivalentes. Por exemplo, uma consulta que vise recuperar o elenco de um determinado time em uma determinada data é escrita de diferentes formas para cada alternativa, mas seu resultado é semanticamente o mesmo em todas as alternativas. As consultas foram analisadas individualmente de acordo com o seu tipo e por fim uma análise sobre o desempenho geral de cada alternativa foi feita.

Os tipos de consultas utilizados para avaliar as alternativas foram: recuperar o elenco corrente dos times, consultas sobre campos atemporais de jogadores, recuperar o elenco dos times em um ponto específico no tempo, selecionar os jogadores que passaram por um determinado time em um determinado período de tempo, atualização no estado corrente de jogadores, atualização de dados atemporais de jogadores e alteração de dados temporais em um determinado período.

Consultas para recuperar o estado corrente das entidades representadas, no caso, de jogadores pertencentes a um determinado time, são importantes porque tendem a ser executadas com frequência. Por isso, é importante que o desempenho na execução dessas consultas seja bom. Além disso, como são consultas utilizadas com frequência, escrevê-las e alterá-las não pode ser excessivamente complexo, sobre pena de

comprometer a manutenabilidade da aplicação de base de dados. Na figura 4.8 abaixo são mostradas as consultas de estado corrente executadas sobre cada uma das alternativas avaliadas.

```
1  -- SOLUTION1
2  SELECT * FROM PLAYER;
3  -- SOLUTION2
4  SELECT * FROM PLAYER;
5  -- SOLUTION3
6  SELECT S.* FROM PLAYER S WHERE S.VALID_SINCE = ( SELECT MAX(VALID_SINCE) FROM PLAYER WHERE
    ID= S.ID AND VALID_SINCE < CURRENT_TIMESTAMP) ORDER BY ID;
7  -- SOLUTION4
8  SELECT * FROM PLAYER WHERE ((VALID_FROM,VALID_TO) OVERLAPS (CURRENT_TIMESTAMP,
   CURRENT_TIMESTAMP) OR (VALID_TO IS NULL AND VALID_FROM < CURRENT_TIMESTAMP));
```

Figura 4.8: Consultas para recuperar o estado corrente de cada jogador nas diferentes alternativas de representação

Na figura 4.8 é possível notar que as consultas das duas primeiras soluções são muito mais simples que as das soluções 3 e 4. Isso acontece porque, além das soluções 1 e 2 separarem dados temporais de dados atemporais, elas mantém, sempre, a informação atual em seus registros, enquanto que nas outras soluções o estado corrente precisa ser obtido através de uma consulta temporal. Ou seja, nas soluções 3 e 4, qualquer consulta sobre algum ponto no tempo será temporal, mesmo que o ponto em questão seja o instante corrente. Assim, as consultas das linhas 2 e 4 na figura 4.8 são essencialmente consultas recuperando todos os dados da tabela atemporal das soluções. Já a consulta da linha 6 requer uma subconsulta que obtém o instante de início de validade imediatamente inferior ao instante corrente. Após a obtenção desse instante, para cada entidade, os dados associados àquele instante são recuperados. Por fim, a consulta da linha 8 utiliza as facilidades oferecidas por um modelo que utiliza períodos para representar a validade temporal pra recuperar os dados do instante atual. O uso do operador "*overlaps*" simplifica bastante essa consulta. Mesmo em bases de dados que não suportam esse operador, é possível utilizar os limites temporais em combinação com símbolos de maior e menor para reproduzir a mesma semântica de maneira simples.

Esses fatores de simplicidade se refletem no desempenho de execução das consultas. A Tabela 4.2 abaixo mostra o tempo, em milissegundos, que cada uma das consultas da figura 4.8 levou pra ser executada sobre as tabelas populadas. Como mencionado anteriormente, cada consulta foi executada seis vezes e uma média das execuções foi computada, pois se verificou experimentalmente que esse número era sufiente para absorver eventuais discrepâncias.

Tabela 4.2: Tempo gasto na execução de consultas de estado corrente, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution 4
1	219	211	2875	1073
2	242	149	2000	904
3	242	168	2800	1057
4	186	191	3500	958
5	151	274	3560	936
6	266	173	2321	988
Média	163	146	2132	740

É possível de se verificar na tabela 4.2 acima que existem alguns picos na variacão do tempo de execução das consultas. Isso provavelmente acontece por processos do sistema operacional rodando em paralelo com os experimentos, concorrendo com os mesmos. Isso se verificou consistentemente em todos os experimentos, e esses picos acabam sendo absorvidos pelas demais execuções que possuem tempos similares, não afetando a análise como um todo.

Na tabela 4.2 é possível verificar que o desempenho das soluções que separam campos temporais e atemporais, nesse tipo de consulta, é muito superior. Já nas soluções 3 e 4, onde a consulta é temporal, é possível notar um desempenho muito superior na solução 4. Isso se deve ao fato de que a consulta da solução 3 envolve uma subconsulta para obter o registro com os valor de time corrente para a entidade, enquanto que na solução 4 um simples teste sobre os limites do período de cada registro basta para determinar qual registro é o que contém os dados atuais. Naturalmente que num contexto onde consultas sobre o estado atual do atributo temporal são muito frequentes, soluções que mantém uma imagem sobre o estado atual são mais indicadas, pois seu desempenho, comparando com soluções onde todos os dados ficam na mesma tabela, chega a ser mais de dez vezes melhor. Desse modo, se as alternativas de tabela separada estudadas não mantivessem o valor corrente do atributo temporal na tabela principal, o desempenho dessas alternativas tenderia a ser mais próximo das outras. Isso é averiguado mais adiante em outro tipo de consulta.

Outro ponto sobre essa otimização é que ela pode ser uma fonte de inconsistências entre a tabela com o atributo temporal e a tabela principal, já que esse campo deve ser mantido atualizado. Isso é especialmente verdadeiro ao se utilizar períodos, quando existe a possibilidade de não haver um registro que seja o atual na tabela de histórico. Essa possibilidade não é considerada nessa análise, mas é uma possibilidade que pode ser real em muitas aplicações.

Da mesma maneira, é importante avaliar se consultas que visam obter informações sobre campos atemporais das tabelas com atributos temporais não são afetadas pela existência dos mesmos. O ideal é que seja possível minimizar o impacto de haverem atributos temporais nas tabelas para consultas que não envolvem o fator tempo. A figura 4.9 mostra como ficam as consultas em cada uma das alternativas de representação.

```
1  --SOLUTION1
2  SELECT NAME FROM PLAYER WHERE ID = %PLAYER.ID%;
3  --SOLUTION2
4  SELECT NAME FROM PLAYER WHERE ID = %PLAYER.ID%;
5  --SOLUTION3
6  SELECT DISTINCT(NAME) FROM PLAYER WHERE ID = %PLAYER.ID%;
7  --SOLUTION4
8  SELECT DISTINCT(NAME) FROM PLAYER WHERE ID = %PLAYER.ID%;
```

Figura 4.9: Consultas para recuperar o nome de cada jogador nas diferentes alternativas de representação

Nesse tipo de consulta, todas as consultas são relativamente simples. O identificador do jogador alvo das consultas é indicado pelo literal “%PLAYER.ID%”. Nas duas primeiras soluções simplesmente se busca o nome do jogador na tabela, uma vez que não há redundância nessa informação. Nas soluções 3 e 4, o problema da redundância se

resolve utilizando uma cláusula "*distinct*" sobre o campo de nome. Mesmo nesse caso, a consulta ainda é extremamente simples. Claro que as consultas das linhas 6 e 8 na figura supõe que as informações de nome são replicadas corretamente sempre que existe uma alteração no atributo temporal, que no caso desse exemplo é o campo "team". A simplicidade dessas consultas se reflete nos números de desempenho, apresentados na Tabela 4.3 abaixo.

Tabela 4.3: Tempo gasto na execução de consultas sobre campos atemporais, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	6	1	15	1
2	0	1	1	1
3	0	1	1	2
4	1	1	1	5
5	1	0	1	1
6	0	0	1	1
Média	1	1	3	1

O desempenho de todas as alternativas nesse tipo de consulta é bastante similar, dado que esse tipo de consulta é simples em qualquer uma delas. Nesse caso, é possível verificar que essas consultas não devem influenciar uma decisão sobre como representar os dados temporais no projeto de uma base de dados.

Já as consultas históricas são uma das razões para haverem atributos temporais na definição das entidades. É necessário que o sistema responda bem quando existe a necessidade de buscar informações sobre as tabelas em um instante ou período específico no tempo. A figura 4.10 mostra como ficam as consultas sobre um instante específico no tempo em cada uma das alternativas apresentadas.

```
1  --SOLUTION1
2  SELECT S.*, P.NAME, P.POSITION, P.NATIONALITY FROM PLAYER_TEAM_HIST S, PLAYER P WHERE S.
   VALID_SINCE = ( SELECT MAX(VALID_SINCE) FROM PLAYER_HIST WHERE ID= S.ID AND VALID_SINCE <
   TIMESTAMP '2008-10-11'  GROUP BY ID ORDER BY ID)  AND P.ID = S.ID ORDER BY ID;
3  --SOLUTION2
4  SELECT * FROM PLAYER_TEAM_HIST A, PLAYER B WHERE A.ID_JOGADOR = B.ID AND ((A.VALID_FROM,A.
   VALID_TO) OVERLAPS (TIMESTAMP '2008-10-11', TIMESTAMP '2008-10-11') OR (A.VALID_TO IS NULL
    AND A.VALID_FROM < TIMESTAMP '2008-10-11'));
5  --SOLUTION3
6  SELECT S.* FROM PLAYER S WHERE S.VALID_SINCE = ( SELECT MAX(VALID_SINCE) FROM PLAYER WHERE
    ID= S.ID AND VALID_SINCE < TIMESTAMP '2008-10-11'  GROUP BY ID ORDER BY ID) ORDER BY ID;
7  --SOLUTION4
8  SELECT * FROM PLAYER WHERE ((VALID_FROM,VALID_TO) OVERLAPS (TIMESTAMP '2008-10-11',
   TIMESTAMP '2008-10-11') OR (VALID_TO IS NULL AND VALID_FROM < TIMESTAMP '2008-10-11'))
```

Figura 4.10: Consultas para recuperar o estado de cada jogador, em um ponto específico no tempo, nas diferentes alternativas de representação

Nas consultas mostradas na figura 4.10, é possível notar que para as soluções 3 e 4 não existe diferença entre essas consultas e as consultas para obter o estado corrente da tabela. Nesse caso, muda apenas o parâmetro referente ao instante desejado, que no caso dessas consultas é o dia 11 de Novembro de 2008. Já nas soluções 1 e 2 é necessário fazer uma consulta temporal, o que para o estado corrente não era necessário. É possível verificar também, que nessas soluções, não apenas se necessita uma consulta temporal, como também existe a necessidade de fazer uma junção entre a tabela com os dados atemporais e a tabela com os dados do atributo temporal. Entretanto, a complexidade

adicionada por essa junção é baixa, sendo a consulta muito similar com a consulta das soluções sem separação entre campos temporais e atemporais.

Em termos de performance, a grande diferença verificada nesse tipo de consulta se dá pelo tipo de representação temporal adotada. Na Tabela 4.4 é possível verificar o resultado da execução dessas consultas.

Tabela 4.4: Tempo gasto na execução de consultas históricas, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	3846	1268	1260	1046
2	2004	1327	2111	1128
3	1824	1405	2123	953
4	1974	1170	2353	876
5	2022	1621	2237	882
6	2024	1395	2107	1053
Média	1712	1023	1524	742

Os resultados apresentados na tabela indicam um desempenho bastante superior de alternativas que utilizam períodos para representar a validade do atributo temporal. Comparando as soluções sem tabela separada, a solução utilizando períodos executou as consultas aproximadamente duas vezes mais rápido em média, enquanto que em soluções com tabela separada a solução utilizando períodos foi aproximadamente 70% mais eficiente. Esses números mostram uma melhora expressiva no desempenho quando períodos são utilizados para representar a validade temporal de um atributo. Em sistemas que apresentem frequentemente consultas de histórico, é mais adequado utilizar períodos para representar os dados temporais.

Outra característica verificada é que a junção entre tabelas temporais e atemporais também causa um impacto razoável sobre o desempenho desse tipo de consulta. Mesmo lidando com um volume de dados maior, soluções com apenas uma tabela foram, em média, 30% mais eficientes nesse tipo de consulta. Cabe ao projetista do sistema determinar se esses 30% compensam a economia de espaço em disco, mostrada na seção anterior, gerada pela separação entre dados temporais e atemporais.

Além de consultas sobre instantes específicos no tempo, muitas vezes as consultas temporais executadas pela aplicação de base de dados são consultas sobre períodos. Essas consultas são frequentemente usadas em relatórios e análises de grandes massas de dados temporais. A figura 4.11 mostra uma consulta sobre períodos em cada uma das alternativas analisadas.

```
1  --SOLUTION1
2  SELECT ID,NAME FROM PLAYER WHERE ID IN( SELECT ID FROM PLAYER_HIST A WHERE TEAM = %TEAM%
   AND VALID_SINCE < TIMESTAMP '01-01-2000 00:00:01' AND 0 = (SELECT COUNT(1) FROM
   PLAYER_HIST WHERE NOT TEAM = %TEAM% AND A.ID = ID AND VALID_SINCE > A.VALID_SINCE AND
   VALID_SINCE < TIMESTAMP '01-01-1997 00:00:01' ) );
3  --SOLUTION2
4  SELECT B.ID, B.NAME FROM PLAYER_HIST A, PLAYER B WHERE A.ID = B.ID AND A.TEAM = %TEAM% AND
    ((A.VALID_FROM,A.VALID_TO) OVERLAPS (TIMESTAMP '01-01-1997 00:00:01', TIMESTAMP
   '01-01-2000 00:00:01') OR (A.VALID_TO IS NULL AND A.VALID_FROM < '01-01-1997 00:00:01'));
5  --SOLUTION3
6  SELECT ID, NAME FROM PLAYER A WHERE TEAM = %TEAM% AND VALID_SINCE < TIMESTAMP '01-01-2000
   00:00:01' AND 0 = (SELECT COUNT(1) FROM PLAYER WHERE NOT TEAM = %TEAM% AND A.ID = ID AND
   VALID_SINCE > A.VALID_SINCE AND VALID_SINCE < TIMESTAMP '01-01-1997 00:00:01' );
7  --SOLUTION4
8  SELECT ID,NAME FROM PLAYER WHERE TEAM = %TEAM% AND (VALID_FROM, VALID_TO) OVERLAPS (
   '01-01-1997 00:00:01',TIMESTAMP '01-01-2000 00:00:01')
```

Figura 4.11: Consultas para recuperar os jogadores que passaram por um determinado time em um determinado período, nas diferentes alternativas de representação

A consulta mostrada na figura 4.11 acima mostra uma consulta para recuperar os jogadores que passaram por um determinado time, representado na figura pelo literal "%TEAM%", em um determinado período, no caso, entre o início de 1997 e o inicio de 2001. É possível notar que em alternativas que apresentam representação de dados temporais com períodos essa consulta é consideravelmente mais simples. Nas alternativas 2 e 4, a consulta é escrita através da simples utilização do operador "*overlaps*". Já nas consultas das alternativas 1 e 3 é necessário uma subconsulta para obter essa mesma semântica. Na Tabela 4.5 é possível verificar o impacto de performance gerado pela necessidade dessa subconsulta.

Tabela 4.5: Tempo gasto na execução de consultas históricas envolvendo períodos, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	807	400	749	380
2	780	453	748	405
3	780	352	717	390
4	764	424	710	430
5	760	374	729	404
6	756	430	775	391
Média	774	405	738	400

É possível verificar, pelos resultados da Tabela 4.5, que nesse tipo de consulta a grande diferença mesmo é em função do uso de instantes ou períodos. As alternativas utilizando períodos tiveram tempo de execução aproximadamente 50% inferior. Por outro lado, pouca diferença se verificou entre alternativas com dados temporais separados ou não.

As consultas analisadas até então foram todas de seleção de dados. Entretanto, também é importante que o sistema possua bom desempenho em atualizações, sejam elas em dados correntes, históricos ou atemporais.

Na maioria dos sistemas a atualização dos dados correntes das entidades representadas é mais frequente que atualizações históricas. Quando esse é o caso, é possível que o projetista escolha uma alternativa de representação que apresenta melhor desempenho em atualizações de dados correntes, mesmo que apresente um desempenho

pior na atualização de dados históricos. A figura 4.12 mostra como ficam consultas desse tipo em cada uma das alternativas analisadas.

```
--SOLUTION1
UPDATE PLAYER SET TEAM = %TEAM% WHERE ID = %PLAYER.ID%;
INSERT INTO PLAYER_TEAM_HIST VALUES (%PLAYER.ID%,%TEAM%,CURRENT_TIMESTAMP);
--SOLUTION2
UPDATE PLAYER SET TEAM = %TEAM% WHERE ID = %PLAYER.ID%;
UPDATE PLAYER_TEAM_HIST VALUES SET VALID_TO = CURRENT_TIMESTAMP WHERE ID = %PLAYER.ID% AND
 VALID_TO IS NULL;
INSERT INTO PLAYER_TEAM_HIST VALUES (%PLAYER.ID%,%TEAM%, CURRENT_TIMESTAMP + INTERVAL '1
DAY', NULL);
--SOLUTION3
INSERT INTO PLAYER VALUES (%PLAYER.ID%,%NAME%,%TEAM%,CURRENT_TIMESTAMP,%POSITION%,%
NATIONALITY%);
--SOLUTION4
UPDATE PLAYER VALUES SET VALID_TO = CURRENT_TIMESTAMP WHERE ID = %DAMIAO.ID% AND VALID_TO
IS NULL
INSERT INTO PLAYER VALUES (%PLAYER.ID%,%NAME%,%TEAM%, CURRENT_TIMESTAMP + INTERVAL '1 DAY'
, NULL,%POSITION%,%NATIONALITY%);
```

Figura 4.12: Operações para atualizar o time de um determinado jogador, nas diferentes alternativas de representação

As consultas apresentadas na figura 4.12 mostram que em apenas uma das soluções apresentadas é possível atualizar dados atuais sem a necessidade de mais de um comando. Na solução 1 é necessário atualizar o time atual do jogador e posteriormente atualizar a informação de histórico referente a essa mudança. O mesmo ocorre na solução 2, mas o fato de ela utilizar um período para determinar a validade da informação faz com que duas atualizações tenham que acontecer, uma no registro que anteriormente continha o dado mais atual (nesse caso representado pelo registro que tem a validade final como “null”) e outra pra inserir o registro que a partir de agora representa o dado atualmente válido. Já a solução 3 é a que apresenta a maneira mais simples de inserir num novo dado como sendo o válido atualmente, basta um único comando referente ao novo time do jogador em questão. Isso se deve ao fato de essa alternativa utilizar apenas um campo para a validade da informação de time. Como explicado no capítulo anterior, ao ser inserido um novo dado os intervalos automaticamente estarão ajustados. Além disso, como não há separação entre tabelas de histórico e tabelas atemporais e de estado corrente, não há a necessidade de uma atualização nesse tipo de tabela, como existe na solução 1. Por fim, a alternativa 4 requer apenas os dois comandos referentes ao uso de períodos: um para atualizar o fim da validade do antigo registro atual e outro para inserir o novo registro atual. Na Tabela 4.6 abaixo é possível verificar como foi o desempenho de cada alternativa nesse tipo de consulta.

Tabela 4.6: Tempo gasto na execução de atualizações de estado corrente, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	2	5	2	39
2	3	10	2	3
3	3	5	1	20
4	15	35	2	27
5	16	6	1	5
6	3	12	2	5
Média	5	9	1	12

Na Tabela 4.6 é possível verificar que a solução 3 tem um desempenho bastante superior ao das outras soluções. É importante ressaltar que mesmo que os números tenham sido baixos para todas as soluções, no exemplo foi feita a atualização do time atual de apenas um jogador. No caso de uma atualização em batch, os resultados verificados seguramente serão potencializados. Assim como a solução 3, a solução 1 também apresentou bom resultado, sendo a segunda mais eficiente. Isso demonstra como, nessa situação, o uso de instantes para representar a validade é consideravelmente mais eficiente. Isso se explica por dois motivos: nesse tipo de representação não é necessário nenhum tipo de procedimento externo para verificar a integridade dos dados já que pode se fazer isso com uma simples chave primária, e é possível executar a atualização com apenas um comando, ao invés de dois como é necessário para soluções que utilizam períodos.

Atualizações históricas não costumam ser tão frequentes, mas também são importantes. De nada adianta ter um sistema que lida com atributos temporais se o seu desempenho quando requisitado a fazer alterações sobre esses dados for ruim. Muitas aplicações sofrem atualizações constantes, portanto é importante que o projetista seja capaz de avaliar qual alternativa de representação de dados temporais se comporta melhor em relação a atualizações quando definir a base de dados. Na figura 4.13 é possível verificar como uma mesma consulta de atualização de dados em um determinado período é escrita em cada uma das alternativas.

```
1  --SOLUTION1
2  UPDATE PLAYER_TEAM_HIST A SET TEAM = 'TEAMB' WHERE TEAM = 'TEAMA' AND VALID_SINCE <
   TIMESTAMP '01-01-2015' AND 0 = (SELECT COUNT(1) FROM PLAYER_TEAM_HIST WHERE NOT TEAM =
   'TEAMA' AND A.ID = ID AND VALID_SINCE > A.VALID_SINCE AND VALID_SINCE > TIMESTAMP
   '01-01-2011' );
3  --SOLUTION2
4  UPDATE PLAYER_TEAM_HIST SET TEAM = 'TEAMB'  WHERE TEAM = 'TEAMA' AND (VALID_FROM,VALID_TO)
    OVERLAPS (TIMESTAMP '01-01-2011',TIMESTAMP '01-01-2015');
5  --SOLUTION3
6  UPDATE PLAYER A SET TEAM = 'TEAMB' WHERE TEAM = 'TEAMA' AND VALID_SINCE < TIMESTAMP
   '01-01-2015' AND 0 = (SELECT COUNT(1) FROM PLAYER WHERE NOT TEAM = 'TEAMA' AND A.ID = ID
   AND VALID_SINCE > A.VALID_SINCE AND VALID_SINCE > TIMESTAMP '01-01-2011' );
7  --SOLUTION4
8  UPDATE PLAYER SET TEAM = 'TEAMB'  WHERE TEAM = 'TEAMA' AND (VALID_FROM,VALID_TO) OVERLAPS
   (TIMESTAMP '01-01-2011',TIMESTAMP '01-01-2015')
```

Figura 4.13: Consultas de atualização de dados em um determinado período, nas diferentes alternativas de representação

As consultas da figura 4.13 ilustram a seguinte situação: alterar o nome do time de “teamA” pra “teamB” em registros cuja validade intersecciona o período entre 2011 e 2015. Nas consultas da figura, é possível verificar que as consultas de alternativas que utilizam períodos são muito mais simples de serem escritas, pelo fato do suporte a verificações de intersecção entre períodos ser nativo nelas. Já as consultas das soluções 1 e 3, que utilizam apenas instantes, dependem de subconsultas que realizam uma junção das tabelas contendo informações temporais com elas mesmas para aplicar essa mesma semântica. Na Tabela 4.7 são mostrados os impactos causados na performance por conta da complexidade adicional das consultas sobre alternativas que utilizam instantes.

Tabela 4.7: Tempo gasto na execução de atualizações temporais, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	5842	401	6342	386
2	6987	401	6073	404
3	5209	406	5499	382
4	6033	433	8216	384
5	5476	445	6462	417
6	5267	385	7475	430
Média	4352	309	5008	300

Os resultados mostrados na Tabela 4.7 evidenciam a dificuldade de soluções baseadas em instantes de lidar com dados que envolvem períodos. Enquanto que a separação entre tabelas temporais e atemporais fez pouca diferença pra esse tipo de consulta, o que era esperado uma vez que mesmo nas soluções onde elas existem, elas não são envolvidas nas consultas, a representação da validade do atributo temporal causou grande impacto, com as soluções que utilizam períodos tendo desempenho mais de quinze vezes superior.

Por fim, o impacto da existência de atributos temporais em atualizações em campos atemporais também deve ser considerado, de acordo com a frequência que elas acontecem. Na figura 4.14 é possível verificar como ficam as consultas desse tipo em cada alternativa.

```
1  --SOLUTION1
2  UPDATE PLAYER SET NAME = '%PLAYERNAME%' WHERE ID = %PLAYER.ID%;
3  --SOLUTION2
4  UPDATE PLAYER SET NAME = '%PLAYERNAME%' WHERE ID = %PLAYER.ID%;
5  --SOLUTION3
6  UPDATE PLAYER SET NAME = '%PLAYERNAME%' WHERE ID = %PLAYER.ID%;
7  --SOLUTION4
8  UPDATE PLAYER SET NAME = '%PLAYERNAME%' WHERE ID = %PLAYER.ID%;
```

Figura 4.14: Consultas de atualização de dados atemporais, nas diferentes alternativas de representação

É possível verificar na figura 4.14 que para atualizar dados atemporais todas as consultas são escritas da mesma forma. Na figura, é mostrada a atualização do nome de um jogador específico, identificado por %PLAYER.ID% para o valor %PLAYERNAME%. O que varia nas alternativas, dependendo do fato de haver ou não separação entre campos temporais e atemporais, é o número de registros afetados. Nas soluções 3 e 4, por haver replicação de dados, as consultas irão afetar um número maior de registros. Entretanto, na Tabela 4.8 abaixo é possível verificar que isso não causa grandes impactos de performance.

Tabela 4.8: Tempo gasto na execução de atualizações temporais, em milissegundos

Run ID/ Solution ID	Solution1	Solution2	Solution3	Solution4
1	3	1	3	3
2	2	2	2	2
3	2	1	3	3
4	1	1	2	3
5	1	1	2	3
6	2	1	3	4
Média	2	1	3	3

Na Tabela 4.8 verifica-se que todas as alternativas tem desempenho similar. Nesse caso, a escolha do modelo causa pouco ou nenhum impacto no desempenho desse tipo de comando.

Claro que cada aplicação tem suas características únicas, que dependem do contexto no qual ela está inserida. Porém, de posse de dados concretos sobre o quanto que cada escolha de modelo impactará o desempenho da aplicação em cada tipo de consulta, o projetista pode tomar uma decisão correta sobre qual alternativa utilizar. Desse modo, ele pode otimizar o desempenho do sistema, evitando um possível retrabalho no caso de haver escolhido uma alternativa de representação que não atende satisfatoriamente as necessidades da aplicação.

Em termos de desempenho, os resultados obtidos nos experimentos permitem visualizar que as alternativas que utilizam períodos para representar a validade de dados temporais tem desempenho bastante superior em consultas que envolvem temporalidade, sem terem o desempenho prejudicado em outros tipos de consultas. Além disso, é possível verificar que separar dados temporais de atemporais oferece economia de espaço em disco, além de possibilitar otimizações como a de armazenar o estado corrente separadamente do histórico, sem causar grande perda de performance em outras consultas. Dito isso, o projetista deveria escolher entre as alternativas 2 e 4 apresentadas, baseando a escolha entre uma delas na sua necessidade de acordo com o contexto da aplicação e as limitações e tipos de consultas mais frequentes que ela apresenta.

4.4.2 Alternativas para representação de entidades temporais

No capítulo anterior foram discutidas duas alternativas para representação da validade temporal de entidades. Assim como com atributos temporais, é possível utilizar instantes ou períodos nessa representação. Entretanto, para entidades não se aplica a alternativa de separar campos temporais de atemporais porque todos os campos são temporais. Naturalmente que se pode manter um histórico separado através de um registro de alterações, mas essa possibilidade, que também foi mostrada no capítulo anterior, é estudada separadamente.

As duas possibilidades de representação de validade temporal mostradas no capítulo anterior foram implementadas e estudadas. Para a análise de todos os aspectos relevantes de representação de dados temporais referentes a entidades, são utilizadas sempre duas entidades temporais em cada uma das duas alternativas, uma das entidades possuindo uma referência para a outra.

Para os experimentos, se utilizou o mesmo contexto dos exemplos da seção sobre entidades temporais do capítulo anterior. Esse contexto apresentava duas entidades presentes em um jogo fictício, “personagem” e “classe de personagem”. Essas duas entidades foram implementadas em ambas as alternativas para representação de dados temporais de entidades. A figura 4.15 mostra o modelo ER e a representação relacional das tabelas da alternativa que utiliza instantes.

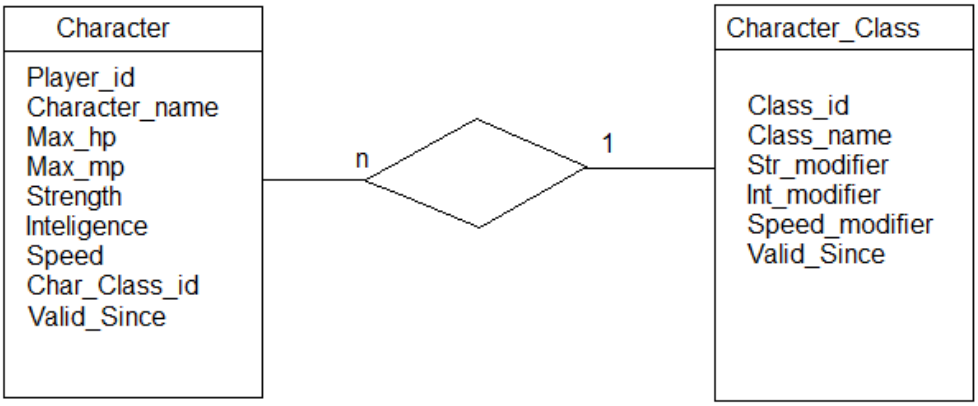


Figura 4.15: Modelo ER e a representação relacional das tabelas da alternativa que utiliza instantes para representar a validade dos dados de entidades temporais

Da mesma maneira, a figura 4.16 mostra o modelo ER e a representação relacional das tabelas da alternativa que utiliza períodos para representar a validade dos dados de entidades temporais.

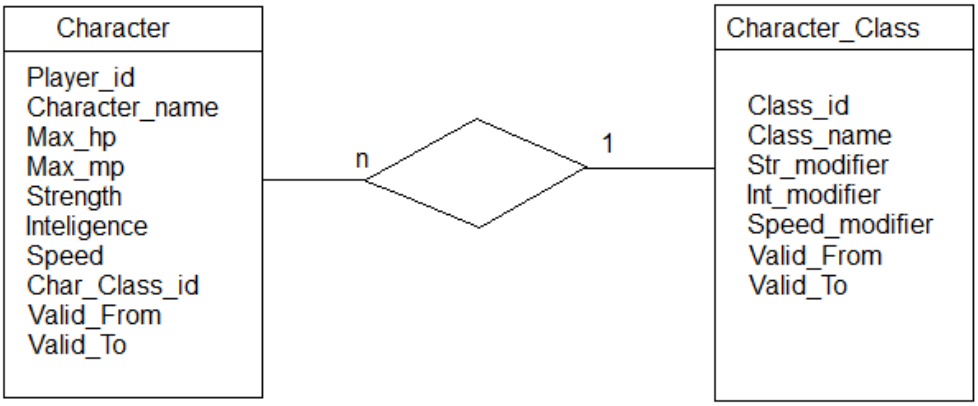


Figura 4.16: Modelo ER e a representação relacional das tabelas da alternativa que utiliza períodos para representar a validade dos dados de entidades temporais

Nos experimentos detalhados a seguir as alternativas são referidas apenas como “utilizando instantes” ou “utilizando períodos” porque como são apenas duas alternativas a serem estudadas, não houve a necessidade de numerá-las.

4.4.2.1Preparação

Da mesma maneira que os experimentos anteriores, sobre atributos temporais, nesses experimentos sobre entidades temporais foi seguido o mesmo ciclo experimental

para análise dos resultados. Primeiramente, os populadores da base de dados que foram escritos para esses experimentos foram configurados para gerar uma quantidade relevante de dados. Nesses experimentos, os parâmetros usados foram: aproximadamente quarenta mil entidades na tabela que faz referência, cada uma das entidades com informações referentes a dois períodos de tempo; pouco mais de quarenta entidades na tabela referenciada, tendo cada uma delas informações referentes a sete períodos de tempo distintos. O número de registros gira em torno de oitenta mil na tabela referenciante e duzentos e oitenta na tabela referenciada.

Apesar do número pouco expressivo em termos de registros na tabela referenciada, os experimentos realizados mostraram que mesmo esse número é suficiente pra apontar diferenças relevantes entre as alternativas no que diz respeito aos critérios utilizados. Na confecção dos experimentos se verificou, inclusive, que um número de registros muito superior a esse na tabela referenciada inviabilizaria a execução dos experimentos no sistema utilizado para esse fim.

Assim como nos experimentos apresentados anteriormente, após a tabela ser populada, foram analisados os números referentes ao volume de dados das tabelas de cada uma das alternativas.

4.4.2.2Análise sobre o Volume de Dados

Após todas as tabelas de cada alternativa serem populadas, o volume de dados em cada uma delas foi analisado. A Tabela 4.9 abaixo mostra as medições de espaço em disco consumido pelas tabelas de cada alternativa.

Tabela 4.9: Volume ocupado por ambas as alternativas para definição de tabelas representando entidades temporais

	Utilizando periodos	Utilizando instantes
Tabela Referenciante	9 MB (0MB índices)	7,2 MB (11MB índices)
Tabela Referenciada	56k (0k índices)	24k (48k índices)

Conforme os dados da Tabela 4.9 acima, excetuando o espaço ocupado por índices não há grandes diferenças em termos de espaço ocupado por cada alternativa. O único fator além dos índices a afetar essa ocupação de espaço em disco é o campo temporal adicional presente nas tabelas que utilizam períodos. Por ser apenas um campo, a diferença de tamanho entre as tabelas é pequena, mesmo quando há um número elevado de registros como na tabela referenciante. Além disso, não há replicação de dados que poderia influenciar o tamanho das tabelas comparando as alternativas.

Por outro lado, o espaço em disco gerado pelo uso de índices, no caso dessas tabelas, é bastante relevante, já que essa acaba sendo a única diferença relevante em termos de espaço em disco. Uma tabela que utilize períodos para representar as informações temporais em geral não vai contar com índices, a menos que o administrador da base de dados os crie explicitamente. Já as tabelas que utilizam instantes usam esses instantes como chave primária, o que faz com que o SGBD automaticamente crie índices sobre esses campos. Dessa forma, essas tabelas tendem a ter um uso de espaço em disco maior para se beneficiarem dos índices nas pesquisas. No fim das contas, o espaço utilizado em tabelas que utilizam instantes tende a ser maior do que as tabelas que utilizam período, com essa diferença de tamanho se tornando mais evidente na medida que a tabela cresce, como é possível verificar pelos números da tabela 4.9.

Por esses motivos, o projetista deve tomar cuidado com os detalhes referentes a cada alternativa quando escolher qual utilizar no seu projeto. Uma análise preliminar poderia levar o projetista a escolher, de forma a priorizar economia em espaço em disco, a alternativa que utiliza instantes, imaginando que por ter um campo temporal a menos iria economizar espaço em disco. Na realidade, ele provavelmente iria acabar ocupando muito mais espaço em disco, por conta dos índices. Em alguns SGBDs, como no caso do PostgreSQL utilizado para esses experimentos, nem existe a opção de desabilitar a criação automática desses índices (POSTGRESQL, 2012).

4.4.2.3Análise sobre a necessidade de verificações de integridade externas

Devido ao problema das referências temporais explicado no capítulo anterior, lidar com entidades temporais que possuem referências entre sí é bastante complicado porque os SGBDs convencionais não oferecem suporte para esse tipo de referência, o que obriga a existência de procedimentos externos que garantam a integridade dos dados.

São necessários três tipos de verificações quando existem referências entre tabelas representando entidades temporais e alguma alteração é feita sobre alguma das tabelas. Primeiramente é necessário garantir que ambas as tabelas possuam chaves primárias temporais. Ou seja, para qualquer ponto no tempo, existe apenas um registro com os valores referentes à entidade para aquele ponto. Na alternativa que utiliza instantes isso é muito simples de se conseguir, basta definir a chave primária da tabela como sendo uma combinação entre o identificador único da entidade presente na tabela e o campo temporal. No caso do exemplo utilizado nesses experimentos, seria uma combinação entre os campos *Player_Id* e *Valid_Since* na tabela *Character* e entre os campos *Class_Id* e *Valid_Since* na tabela *Character_Class*. Já na alternativa que utiliza períodos, isso não é suficiente para garantir a existência de uma chave primária temporal, e se faz presente a necessidade de que isso seja garantido por um procedimento externo. Nas alternativas estudadas, todos esses procedimentos foram implementados utilizando *triggers*. Na figura 4.17 são mostrados os comandos usados para criar as *triggers* necessárias.

```
1  -- TABELA REFERENCIANTE
2  CREATE TRIGGER char_pk_trig
3    BEFORE INSERT OR UPDATE
4    ON CHARACTER
5    FOR EACH ROW
6    EXECUTE PROCEDURE char_base_primary_key();
7
8  -- TABELA REFERENCIADA
9  CREATE TRIGGER char_class_pk_trig
10   BEFORE INSERT OR UPDATE
11   ON CHARACTER_CLASS
12   FOR EACH ROW
13   EXECUTE PROCEDURE char_class_primary_key();
```

Figura 4.17: Declaração das *triggers* que garantem a existência de uma chave primária temporal nas tabelas que utilizam períodos

Essas *triggers* indicam quais procedimentos serão chamados para responder aos eventos de alteração de dados em cada tabela. No caso da figura, os procedimentos *char_base_primary_key* (tabela referenciante) e *char_class_primary_key* (tabela referenciada) serão chamados no caso de alterações (inserções ou atualizações) nas

tabelas *Character* e *Character_Class*. Na figura 4.18 abaixo é possível visualizar a implementação do procedimento *char_base_primary_key* em PgPL-SQL.

```
BEGIN
FOR R IN SELECT * FROM CHARACTER WHERE PLAYER_ID = NEW.PLAYER_ID LOOP
    IF (NEW.VALID_FROM,NEW.VALID_TO) OVERLAPS (R.VALID_FROM,R.VALID_TO) THEN
        RAISE EXCEPTION 'INTERVAL CONFLICT! ID= %, VALID_FROM=%, VALID_TO=%', NEW.
        PLAYER_ID, NEW.VALID_FROM, NEW.VALID_TO;
    END IF;
END LOOP;
RETURN NEW;
END;
```

Figura 4.18: Implementação de procedimento de verificação de chaves primárias temporais na tabela referenciante

Esse procedimento basicamente checa o registro a ser inserido ou atualizado contra todos os registros com o mesmo identificador, para verificar que não existe sobreposição entre os intervalos de validade de registros referentes à mesma entidade. O procedimento para a tabela *Character_Class* é basicamente o mesmo, mudando apenas a referência da tabela e o nome do campo de identificador, como mostrado na figura 4.19.

```
BEGIN
FOR R IN SELECT * FROM CHARACTER_CLASS WHERE CLASS_ID = NEW.PLAYER_ID LOOP
    IF (NEW.VALID_FROM,NEW.VALID_TO) OVERLAPS (R.VALID_FROM,R.VALID_TO) THEN
        RAISE EXCEPTION 'INTERVAL CONFLICT! ID= %, VALID_FROM=%, VALID_TO=%', NEW.
        PLAYER_ID, NEW.VALID_FROM, NEW.VALID_TO;
    END IF;
END LOOP;
RETURN NEW;
END;
```

Figura 4.19: Implementação de procedimento de verificação de chaves primárias temporais na tabela referenciada

A segunda verificação necessária quando existem referências entre entidades temporais é a garantia que em tabelas referenciantes qualquer registro inserido ou atualizado possui correspondência da entidade que ele referencia, na tabela referenciada, durante todo o seu período de validade. Ou seja, a chave estrangeira indicada na tabela referenciante é satisfeita em qualquer ponto no tempo. Naturalmente, SGBDs convencionais não oferecem construtores para esse tipo de restrição, o que requer a existência de procedimentos externos para fazer essa verificação. Assim como nos procedimentos anteriores, para a realização desses experimentos isso foi feito utilizando *triggers*, como mostrado na figura 4.20 abaixo.

```
-- UTILIZANDO INSTANTES
  CREATE TRIGGER char_base_inst_fk_trig
  BEFORE INSERT OR UPDATE
  ON CHARACTER
  FOR EACH ROW
  EXECUTE PROCEDURE char_base_inst_fk()

-- UTILIZANDO PERIODOS
  CREATE TRIGGER char_base_per_trig
  BEFORE INSERT OR UPDATE
  ON CHARACTER
  FOR EACH ROW
  EXECUTE PROCEDURE char_base_per_fk();
```

Figura 4.20: Declaração das *triggers* que garantem que os registros inseridos na tabela referenciante tem a sua referência satisfeita na tabela referenciada durante toda a sua validade

O comando para criar as *triggers* dessa verificação é igual em ambas as alternativas. O que varia é a implementação do procedimento que verifica a existência da correspondência na tabela referenciada. A implementação desse procedimento na alternativa que utiliza instantes pode ser visualizada na figura 4.21 abaixo.

```
BEGIN
    IF NOT EXISTS (
        SELECT * FROM CHARACTER_CLASS WHERE VALID_SINCE <= NEW.VALID_SINCE AND NEW.
        CHAR_CLASS_ID = CLASS_ID
    )
    THEN
    RAISE EXCEPTION 'FK IS NOT SATISFIED IN SOME POINT IN CHANGED ROW VALIDITY TIME';
        END IF;
    RETURN NEW;
END;
```

Figura 4.21: Procedimento que garante a integridade das referências provenientes da tabela referenciante no momento de uma inserção ou atualização de dados da mesma, na alternativa que utiliza instantes

Como é possível perceber pela figura 4.21, o procedimento que garante a integridade das referências existentes na tabela referenciada no momento de uma inserção ou atualização de dados da mesma, na alternativa que utiliza instantes, é bastante simples. Ele simplesmente verifica se para o novo registro que está sendo inserido ou atualizado na tabela, se a classe de personagem que ele referencia através do campo "*Char_Class_Id*" possui algum registro com valor inicial de validade anterior ao seu. Por definição, nesse modelo de dados um registro é valido do início da sua validade, indicada no campo *Valid_Since* até que exista um outro registro com *Valid_Since* mais atual ou sem validade final definida, sendo nesse caso o registro válido atualmente. Por isso, basta verificar que existe um registro na tabela de classes mais antigo que o registro sendo inserido na tabela de personagens, que a restrição de integridade referente à chave estrangeira temporal estará sendo satisfeita.

Já na alternativa que utiliza períodos, o procedimento que implementa essa mesma verificação é bem mais complicado. Isso acontece, principalmente, porque o conceito em sí nesse caso é bem mais complexo. Ao invés de existir apenas a necessidade da existência de um único registro na tabela referenciada com validade iniciando em um ponto no tempo igual ou anterior à validade do registro sendo inserido no registro da tabela referenciante, nessa alternativa é preciso garantir que um registro, ou uma união de registros, cubra todo o intervalo de validade do registro sendo inserido. Isso é

complexo porque envolve combinar diferentes registros ao se fazer essa verificação, o que acaba sendo extremamente complexo. Na figura 4.22 é apresentado o procedimento responsável por fazer essa verificação de satisfação da referência de chave estrangeira.

```
1  BEGIN
2      IF NOT EXISTS (
3      -- BLOCO 1
4      SELECT *
5      FROM CHARACTER_CLASS AS P
6      WHERE NEW.CHARACTER_CLASS_ID = P.CLASS_ID
7      AND P.VALID_FROM <= NEW.VALID_FROM
8      AND NEW.VALID_FROM < P.VALID_TO)
9      -- BLOCO 2
10     OR NOT EXISTS (
11     SELECT *
12     FROM CHARACTER_CLASS AS P
13     WHERE NEW.CHARACTER_CLASS_ID = P.CLASS_ID
14     AND P.VALID_FROM < NEW.VALID_TO
15     AND NEW.VALID_TO <= P.VALID_TO)
16     -- BLOCO 3
17     OR EXISTS (
18     SELECT *
19     FROM CHARACTER_CLASS AS P
20     WHERE NEW.CHARACTER_CLASS_ID = P.CLASS_ID
21     AND NEW.VALID_FROM < P.VALID_TO
22     AND P.VALID_TO < NEW.VALID_TO
23     AND NOT EXISTS (
24     SELECT *
25     FROM CHARACTER_CLASS AS P2
26     WHERE P2.CLASS_ID = P.CLASS_ID
27     AND P2.VALID_FROM <= P.VALID_TO + '1 DAY'
28     AND P.VALID_TO < P2.VALID_TO))
29     THEN
30     RAISE EXCEPTION 'FK IS NOT SATISFIED IN SOME POINT IN CHANGED ROW VALIDITY TIME';
31         END IF;
32     RETURN NEW;
33 END;
```

Figura 4.22: Procedimento que garante a integridade das referências provenientes da tabela referenciante no momento de uma inserção ou atualização de dados da mesma, na alternativa que utiliza períodos

Como mencionado anteriormente, o procedimento da figura 4.22 é bem mais complexo que o da figura 4.21, seu equivalente da alternativa que utiliza instantes. Esse procedimento é dividido em três grandes blocos, cujos inicios estão marcados na figura através de comentários nas linhas 3, 9 e 16. O primeiro bloco garante que existe algum registro na tabela referenciada que cobre o limite inferior da validade do registro sendo inserido na tabela referenciante. O segundo bloco cobre o limite superior desse mesmo registro. Por fim, o terceiro bloco verifica se há, dentro da validade do registro sendo inserido, algum registro que não é imediatamente seguido por outro registro referente a mesma entidade, ou seja, esse bloco procura por falhas temporais dentro da validade do registro que está sendo inserido na tabela referenciante. Basicamente, esse procedimento garante que os limites da validade do novo registro da tabela referenciante estão cobertos, e que não existem falhas temporais dentro da validade desse novo registro. Se alguma dessas condições não for satisfeita, a inserção falha. Uma observação importante é que esse procedimento considera a granularidade dos dados como sendo em dias. Para utilizar uma granularidade mais específica, deve ser alterado o literal da linha 27 (“1 day”) para a granularidade desejada, como por exemplo “1 second”.

Esse procedimento naturalmente envolve um custo de processamento bastante superior se comparado ao procedimento da alternativa que utiliza instantes. Na seção seguinte esses impactos são quantificados.

Já a terceira e igualmente importante verificação que se deve fazer com relação a chaves estrangeiras temporais é a verificação se as referencias de chave estrangeira temporal são mantidas após alterações na própria tabela referenciada. Basicamente, quando um ou mais registros são alterados ou removidos dessa tabela, é necessário que seja feita uma verificação de se as referências de chave estrangeira temporal continuam sendo satisfeitas. Primeiramente, é necessário definir a *trigger* que irá disparar o procedimento que irá fazer essa verificação. Na figura 4.23 abaixo são mostradas as definições das *triggers* que irão disparar os procedimentos necessários.

```
1  -- UTILIZANDO INSTANTES
2  CREATE TRIGGER char_class_per_referecial_int_trig
3    AFTER UPDATE OR DELETE
4    ON CHARACTER_CLASS
5    FOR EACH ROW
6    EXECUTE PROCEDURE char_class_per_referecial_int();
7
8  -- UTILIZANDO PERIODOS
9  CREATE TRIGGER char_class_inst_referecial_int_trig
10   AFTER UPDATE OR DELETE
11   ON CHARACTER_CLASS
12   FOR EACH ROW
13   EXECUTE PROCEDURE char_class_inst_referecial_int();
```

Figura 4.23: Declaração das *triggers* que garantem que os registros removidos ou alterados na tabela referenciada não afetam referências provenientes da tabela referenciante

Novamente, o comando é bastante similar aos já empregados nas definições de *triggers* anteriores, com a diferença que agora o comando é executado sobre o estado da tabela após as alterações (diretriz *AFTER*, linhas 3 e 10). Já os procedimentos invocados por essas *triggers* são bastante similares aos executados na verificação apresentada anteriormente, com pequenas mudanças. Na figura 4.24 é possível verificar como fica o procedimento que faz essa verificação na alternativa que utiliza instantes.

```
1  BEGIN
2      IF EXISTS
3          (
4          SELECT * FROM CHAR_BASE_POINT_BASED AS I
5              WHERE
6              CHAR_CLASS_ID = OLD.ID AND
7              NOT EXISTS (
8                  SELECT * FROM CHAR_CLASS_POINT_BASED WHERE VALID_SINCE <= I.VALID_SINCE
                   AND I.CHAR_CLASS_ID = CLASS_ID
9              )
10         )
11         THEN RAISE EXCEPTION 'REFERENCIAL INTEGRITY VIOLATED, CANNOT CHANGE THE TABLE';
12     END IF;
13     RETURN NULL;
14 END;
```

Figura 4.24: Procedimento que garante a integridade das referências existentes para a tabela referenciada no momento de uma inserção ou atualização de dados da tabela referenciante, na alternativa que utiliza instantes

É possível verificar que o código mostrado na figura 4.24 é bastante similar ao que verifica as referências existentes quando uma mudança ocorre na tabela referenciante. Existe, entretanto, uma diferença crucial entre eles: nesse código, todos os registros da tabela referenciante que podem vir a ser afetados por uma mudança gerada em um registro da tabela referenciada são testados em relação às suas referências para a tabela referenciada. Além disso, comandos de atualização e remoção de dados podem afetar muitos registros ao mesmo tempo, fazendo com que esse procedimento rode muitas vezes. Isso tem um peso considerável na performance de operações desse tipo nas alternativas estudadas. Na subseção seguinte, esses impactos são estudados mais a fundo.

Da mesma maneira, o procedimento que efetua a mesma verificação na alternativa que utiliza períodos funciona similarmente ao procedimento da figura 4.24. Ele é basicamente o mesmo procedimento apresentado anteriormente para garantir a integridade das referências temporais quando um registro é inserido na tabela referenciante, mas executado para cada registro que possa ser alterado pela alteração na tabela. Na figura 4.25 é apresentado o código que executa essa tarefa.

```
1  BEGIN
2      IF EXISTS
3      (
4          SELECT *
5          FROM CHAR_BASE AS I
6          WHERE CHAR_CLASS_ID = OLD.CLASS_ID
7          AND(
8              NOT EXISTS (
9                  SELECT *
10                 FROM CHAR_CLASS AS P
11                 WHERE I.CHAR_CLASS_ID = P.CLASS_ID
12                 AND P.VALID_FROM <= I.VALID_FROM
13                 AND I.VALID_FROM < P.VALID_TO)
14             OR NOT EXISTS (
15                 SELECT *
16                 FROM CHAR_CLASS AS P
17                 WHERE I.CHAR_CLASS_ID = P.CLASS_ID
18                 AND P.VALID_FROM < I.VALID_TO
19                 AND I.VALID_TO <= P.VALID_TO)
20             OR EXISTS (
21                 SELECT *
22                 FROM CHAR_CLASS AS P
23                 WHERE I.CHAR_CLASS_ID = P.CLASS_ID
24                 AND I.VALID_FROM < P.VALID_TO
25                 AND P.VALID_TO < I.VALID_TO
26             AND NOT EXISTS (
27                 SELECT *
28                 FROM CHAR_CLASS AS P2 -- HAS A GAP
29                 WHERE P2.CLASS_ID = P.CLASS_ID
30                 AND P2.VALID_FROM <= P.VALID_TO + '1 DAY'
31                 AND P.VALID_TO < P2.VALID_TO))
32         )
33     )
34     THEN RAISE EXCEPTION 'REFERENCIAL INTEGRITY VIOLATED, CANNOT CHANGE THE TABLE';
35     END IF;
36     RETURN NULL;
37 END;
```

Figura 4.25: Procedimento que garante a integridade das referências existentes para a tabela referenciada no momento de uma inserção ou atualização de dados da tabela referenciante, na alternativa que utiliza períodos

A consulta mais interna da figura 4.25, que vai da linha 7 até a linha 32, é rigorosamente a mesma da figura 4.22. A diferença é que nessa situação, ela é precedida

por uma seleção de registros que podem ser afetados por alterações nas informações da entidade na tabela referenciada, e executada para cada um desses registros. Além disso, cada registro alterado na tabela referenciada dispara uma execução desse procedimento.

Devido à simplicidade consideravelmente menor dos procedimentos das figuras 4.22 e 4.25 em relação aos procedimentos das figuras 4.21 e 4.24, a alternativa que utiliza períodos tende a ter um desempenho muito pior em termos de assegurar a integridade das referencias de chave estrangeira temporal entre as tabelas. O impacto disso nos comandos que alteram dados nas tabelas pode ser verificado na subseção seguinte. Além disso, a simplicidade menor desses procedimentos também torna sua manutenção mais complicada, dificultando a implementação e validação de alterações sobre eles. Em termos da necessidade de procedimentos para verificação de integridade de dados, a alternativa que utiliza instantes tende a ter desempenho bastante superior e facilita bastante a implementação por parte do desenvolvedor da aplicação de base de dados.

4.4.2.4Execução e Análise de Simplicidade de Consultas

Para avaliar o desempenho de cada uma das alternativas estudadas, foi construído um conjunto de consultas, avaliando como cada alternativa se comporta com os diferentes tipos de consultas e comandos que podem ser executados sobre suas tabelas. As consultas de cada tipo construídas para cada alternativa são semanticamente equivalentes entre si.

Para possibilitar uma avaliação consistente do desempenho de cada alternativa, foram definidos os seguintes tipos de consultas: consultas de estado corrente cruzando as tabelas, consultas de histórico cruzando ambas as tabelas, consultas temporais e atemporais complexas, inserções na tabela referenciante e remoções e atualizações na tabela referenciada. Com esse conjunto de consultas é possível cobrir a maior parte das situações possíveis de serem encontradas em uma aplicação de base de dados que modele entidades temporais com referências entre elas.

Assim como com atributos temporais, em aplicações que envolvem entidades temporais existe a necessidade de existirem consultas sobre o estado corrente das entidades. Como esse é um tipo de consulta que tende a ser frequente, é necessário que o desempenho desse tipo de consulta seja bom. Para avaliar esse desempenho, foram criadas duas consultas referentes a estado corrente. Uma que busca o valor de uma propriedade da tabela referenciada para uma entidade da tabela referenciante, e outra que busca entidades na tabela referenciada com base no estado da tabela referenciante. Nas figuras 4.26 e 4.27 abaixo, é mostrada a implementação dessas consultas em ambas as alternativas estudadas.

```
1  --BUSCA DE VALOR REFERENTE A UMA ENTIDADE DA TABELA REFERENCIANTE NA TABELA REFERENCIADA
2  --UTILIZANDO INSTANTES
3  SELECT STR_MODIFIER FROM CHARACTER,CHARACTER_CLASS WHERE CHARACTER_NAME = %CHARACTER_NAME
   % AND CHARACTER_CLASS.CLASS_ID = CHARACTER.CHAR_CLASS_ID AND (CHARACTER_CLASS.VALID_SINCE
    = (SELECT MAX(VALID_SINCE) FROM CHARACTER_CLASS WHERE VALID_SINCE < CURRENT_TIMESTAMP))
   AND (CHARACTER.VALID_SINCE = (SELECT MAX(VALID_SINCE) FROM CHARACTER WHERE VALID_SINCE <
   CURRENT_TIMESTAMP))
4
5  --UTILIZANDO PERÍODOS
6  SELECT STR_MODIFIER FROM CHARACTER,CHARACTER_CLASS WHERE CHARACTER_NAME = %CHARACTER_NAME
   % AND CHARACTER_CLASS.CLASS_ID = CHARACTER.CHAR_CLASS_ID AND (CHARACTER_CLASS.VALID_FROM,
   CHARACTER_CLASS.VALID_TO) OVERLAPS (CURRENT_TIMESTAMP,CURRENT_TIMESTAMP) AND (CHARACTER.
   VALID_FROM,CHARACTER.VALID_TO) OVERLAPS (CURRENT_TIMESTAMP,CURRENT_TIMESTAMP)
```

Figura 4.26: Implementação de uma consulta de estado corrente em ambas as alternativas estudadas

A primeira consulta, presente na figura 4.26, procura pelo valor de *"Str_Modifier"*, um campo da tabela referenciada, referente à entidade identificada pelo nome *Character_Name*. Na consulta, o marcador *%CHARACTER_NAME%* indica onde é feita a comparação que busca a entidade com esse nome. Em outras palavras, essa consulta busca o modificador de força da classe do personagem cujo nome substitui *%CHARACTER_NAME%* na consulta. Essa é uma consulta que cruza dados de ambas as tabelas e que no contexto utilizado para os experimentos seguramente seria uma consulta bastante frequente.

A escrita dessa consulta é relativamente mais simples na alternativa que utiliza períodos porque o período de validade de cada tupla é bem definido na própria tupla, enquanto que na alternativa que utiliza instantes a validade de qualquer registro sempre depende dos outros registros da tabela. Por isso, saber qual o registro corrente na alternativa que utiliza períodos é mais simples. Mesmo assim, em termos de desempenho não há uma grande diferença entre as alternativas, conforme mostrado na Tabela 4.10 abaixo.

Tabela 4.10: Tempo gasto na primeira consulta de estado corrente, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	94	110
2	81	47
3	78	47
4	74	47
5	93	47
6	75	62
Média	83	60

A pouca diferença entre o despenho das alternativas nessa primeira consulta se deve à grande simplicidade da consulta. Com a existência do parâmetro referente ao nome do personagem na consulta, a busca pela entidade na tabela referenciada é feita através da chave estrangeira temporal, e isso faz com que poucas operações temporais sejam necessárias para resolver a consulta, e é justamente nessas operações que a alternativa que utiliza instantes teria dificuldades adicionais. Na consulta da figura 4.27 abaixo, entretanto, o cenário é diferente. Ela busca as classes das quais atualmente nenhum personagem faz parte.

```
1  --BUSCA DE ENTIDADES NA TABELA REFERENCIADA DE ACORDO COM O ESTADO DA TABELA REFERENCIANTE
2  --UTILIZANDO INSTANTES
3  SELECT CHARACTER_CLASS.CLASS_NAME FROM CHARACTER_CLASS WHERE NOT EXISTS ( SELECT * FROM
   CHARACTER WHERE CHARACTER.CHAR_CLASS_ID = CHARACTER_CLASS.CLASS_ID AND (CHARACTER.
   VALID_SINCE = (SELECT MAX(VALID_SINCE) FROM CHARACTER WHERE VALID_SINCE <
   CURRENT_TIMESTAMP))) AND (CHARACTER_CLASS.VALID_SINCE = (SELECT MAX(VALID_SINCE) FROM
   CHARACTER_CLASS WHERE VALID_SINCE < CURRENT_TIMESTAMP))
4
5
6  --UTILIZANDO PERÍODOS
7  SELECT CHARACTER_CLASS.CLASS_NAME FROM CHARACTER_CLASS WHERE NOT EXISTS ( SELECT * FROM
   CHARACTER WHERE CHARACTER.CHAR_CLASS_ID = CHARACTER_CLASS.CLASS_ID AND (CHARACTER.
   VALID_FROM,CHARACTER.VALID_TO) OVERLAPS (CURRENT_TIMESTAMP,CURRENT_TIMESTAMP)) AND (
   CHARACTER_CLASS.VALID_FROM,CHARACTER_CLASS.VALID_TO) OVERLAPS (CURRENT_TIMESTAMP,
   CURRENT_TIMESTAMP)
```

Figura 4.27: Implementação de outra consulta de estado corrente em ambas as alternativas estudadas

Apesar de parecer muito semelhante à consulta apresentada na figura 4.26, a consulta da figura 4.27 tem uma diferença importante. A ausência de um parâmetro fixo referente a um campo atemporal, que limitaria a quantidade de operações temporais, acaba fazendo grande diferença aqui. Nessa consulta, se verifica a não existência de um registro de personagem ligado às classes existentes na tabela referenciada. Para isso é primeiro necessário verificar quais classes existem atualmente, e se para cada uma delas existe ao menos um registro de personagem existente atualmente. Com essas condições, o número de operações temporais aumenta consideravelmente, e o impacto disso fica evidente na Tabela 4.11 abaixo.

Tabela 4.11: Tempo gasto na segunda consulta de estado corrente, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	757	12
2	149	15
3	760	13
4	771	15
5	150	15
6	153	14
Média	457	14

A solução utilizando instantes teve o seu tempo médio de execução bastante piorado nessa consulta. Como mencionado anteriormente, a dificuldade maior desse modelo em lidar com determinados tipos de operações temporais acaba fazendo a diferença, sendo o tempo de execução da consulta nesse modelo muitas vezes pior. Já na alternativa que utiliza períodos, o tempo é até menor que o da consulta anterior. Isso acontece porque diferentemente da consulta anterior, nessa consulta não é necessária a junção implícita que existe na consulta anterior, a junção só é feita quando é encontrada uma classe que existe no ponto no tempo que representa o instante atual.

Assim como consultas sobre o estado corrente das tabelas, é necessário que exista um bom desempenho em consultas temporais, afinal um dos motivos de se modelar entidades temporais é justamente para que seja possível fazer consultas históricas sobre as entidades. Para avaliar o desempenho desse tipo de consulta foi utilizada uma consulta que cruza dados de ambas as tabelas para obter uma informação específica de uma entidade em um determinado ponto no tempo. Na figura 4.28 é possível ver a implementação dessa consulta em ambas as alternativas estudadas.

```
1  -- UTILIZANDO INSTANTES
2  SELECT CLASS_NAME FROM CHARACTER, CHARACTER_CLASS  WHERE CHARACTER.CHARACTER_NAME = %
   CHARACTER_NAME% AND CHARACTER.CHAR_CLASS_ID = CHARACTER_CLASS.CLASS_ID AND (
   CHARACTER_CLASS.VALID_SINCE = (SELECT MAX(VALID_SINCE) FROM CHARACTER_CLASS WHERE
   VALID_SINCE < %INSTANT%)) AND (CHARACTER.VALID_SINCE = (SELECT MAX(VALID_SINCE) FROM
   CHARACTER WHERE VALID_SINCE < %INSTANT%))
3
4  -- UTILIZANDO PERÌODOS
5  SELECT CLASS_NAME FROM CHARACTER, CHARACTER_CLASS  WHERE CHARACTER.CHARACTER_NAME = %
   CHARACTER_NAME% AND CHARACTER.CHAR_CLASS_ID = CHARACTER_CLASS.CLASS_ID AND (
   CHARACTER_CLASS.VALID_FROM,CHARACTER_CLASS.VALID_TO) OVERLAPS (%INSTANT%,%INSTANT%) AND (
   CHARACTER.VALID_FROM,CHARACTER.VALID_TO) OVERLAPS (%INSTANT%,%INSTANT%)
```

Figura 4.28: Implementação de consulta de histórico em ambas as alternativas estudadas

A consulta da figura 4.28 busca saber qual a classe de um personagem específico em um determinado ponto no tempo. O personagem é indicado por *%CHARACTER_NAME%* na figura, enquanto que o instante para o qual se busca a informação é identificado por *%INSTANT%* .

É possível verificar que a consulta é bastante semelhante à consulta da figura 4.26, mudando apenas o campo buscado na tabela referenciada. Isso acontece porque quando as consultas temporais são sobre entidades temporais, qualquer consulta, seja de estado corrente ou não, é temporal, mudando apenas o parâmetro que indica qual é o instante no tempo para o qual se estão buscando as informações. Logo, não existem diferenças entre consultas de estado corrente ou de histórico quando se envolvem entidades temporais. Isso é possível de se verificar na Tabela 4.12 abaixo, que mostra resultados bastante semelhantes ao da Tabela 4.10.

Tabela 4.12: Tempo gasto na consulta de histórico, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	72	41
2	69	47
3	69	46
4	71	47
5	71	47
6	69	78
Média	70	51

Como esperado, na Tabela 4.12 os números são bastante semelhantes aos números da Tabela 4.10, com apenas uma pequena variação.

As consultas utilizadas até agora foram consultas mais simples, básicas, que a aplicação de base de dados necessita executar com bom desempenho, mas que são fáceis de escrever. É necessário verificar também qual a dificuldade existente e o desempenho ao se executar consultas mais complexas, com condições mais elaboradas do que as vistas até então. Para fazer essa verificação foram implementadas três consultas, cada uma com um condicional diferente, com a intenção de verificar qual a dificuldade para se escrever cada uma delas e como cada alternativa se comporta ao executá-las. Essas consultas são mostradas nas figuras 4.29, 4.30 e 4.31 abaixo.

```
1  -- UTILIZANDO INSTANTES
2  SELECT ID,CHARACTER_NAME FROM CHARACTER CB1 , CHARACTER_CLASS CC1 WHERE CB1.CHAR_CLASS_ID
   = CC1.CLASS_ID AND CC1.CLASS_NAME = %CHAR_NAME1% AND (CC1.VALID_SINCE = (SELECT MAX(
   VALID_SINCE) FROM CHARACTER_CLASS WHERE VALID_SINCE < %INSTANT%)) AND (CB1.VALID_SINCE = (
   SELECT MAX(VALID_SINCE) FROM CHARACTER WHERE VALID_SINCE < %INSTANT%)) AND NOT EXISTS (
   SELECT CHARACTER_NAME FROM CHARACTER CB2 , CHARACTER_CLASS CC2 WHERE CB1.ID = CB2.ID AND
   CB2.CHAR_CLASS_ID = CC2.CLASS_ID AND CC2.CLASS_NAME = %CHAR_NAME2% )
3
4  -- UTILIZANDO PERÍODOS
5  SELECT ID,CHARACTER_NAME FROM CHARACTER CB1 , CHARACTER_CLASS CC1 WHERE CB1.CHAR_CLASS_ID
   = CC1.CLASS_ID AND CC1.CLASS_NAME = %CHAR_NAME1% AND (CC1.VALID_FROM,CC1.VALID_TO)
   OVERLAPS (%INSTANT%,%INSTANT%) AND (CB1.VALID_FROM,CB1.VALID_TO) OVERLAPS (%INSTANT%,%
   INSTANT%) AND NOT EXISTS ( SELECT CHARACTER_NAME FROM CHARACTER CB2 , CHARACTER_CLASS CC2
   WHERE CB1.ID = CB2.ID AND CB2.CHAR_CLASS_ID = CC2.CLASS_ID AND CC2.CLASS_NAME = %
   CHAR_NAME2%)
```

Figura 4.29: Implementação de consulta complexa envolvendo ambas as tabelas, em cada uma das alternativas estudadas

A consulta da figura 4.29 implementa o seguinte condicional: listar personagens cujo nome foi *%CHAR_NAME1%* no instante *%INSTANT%* mas que nunca tenham sido *%CHAR_NAME2%* em nenhum ponto no passado. Essa é uma consulta que depende de três parâmetros e envolve verificações temporais, pelo menos uma para cada entidade. Entretanto, o fato de haver uma comparação com um campo textual juntamente com a primeira verificação de tempo, diminui a importância do tempo na consulta. Além disso, por se tratar de uma consulta que busca informações relacionadas principalmente a instantes, a simplicidade de escrita dela em cada uma das alternativas é bastante similar. É possível verificar como essas características refletem no desempenho, mostrado na Tabela 4.13 abaixo.

Tabela 4.13: Tempo gasto na segunda consulta de estado corrente, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	134	104
2	130	94
3	124	109
4	126	110
5	127	125
6	127	109
Média	128	109

A alternativa utilizando períodos se sai levemente melhor, de acordo com as informações da Tabela 4.13. Isso não é surpreendente, pois, como discutido antes, a consulta acaba sendo bastante similar em ambas as alternativas. Algo parecido se verifica na consulta da figura 4.30 abaixo.

```
1  -- UTILIZANDO INSTANTES
2  SELECT S2.CLASS_ID, S2.VALID_SINCE AS RAISE_DATE FROM CHARACTER_CLASS AS S1,
   CHARACTER_CLASS AS S2 WHERE S2.STR_MODIFIER > S1.STR_MODIFIER AND S1.CLASS_ID = S2.
   CLASS_ID AND S1.VALID_SINCE < S2.VALID_SINCE AND NOT EXISTS ( SELECT * FROM
   CHARACTER_CLASS WHERE CLASS_ID=S1.CLASS_ID AND VALID_SINCE > S1.VALID_SINCE AND
   VALID_SINCE < S2.VALID_SINCE )
3
4  -- UTILIZANDO PERÍODOS
5  SELECT S2.CLASS_ID, S2.VALID_FROM AS RAISE_DATE FROM CHARACTER_CLASS AS S1,
   CHARACTER_CLASS AS S2 WHERE S2.STR_MODIFIER > S1.STR_MODIFIER AND S1.CLASS_ID = S2.
   CLASS_ID AND S1.VALID_TO < S2.VALID_FROM AND NOT EXISTS ( SELECT * FROM CHARACTER_CLASS
   WHERE CLASS_ID=S1.CLASS_ID AND VALID_FROM > S1.VALID_TO AND VALID_TO < S2.VALID_TO )
```

Figura 4.30: Implementação de outra consulta complexa envolvendo ambas as tabelas, em cada uma das alternativas estudadas

A consulta apresentada na figura 4.30 visa descobrir quais as datas em que o modificador de força (campo "*Str_Modifier*") de alguma classe aumentou. Essa consulta, apesar de ter uma semântica complexa e de sua escrita envolver uma série de verificações, é similar em ambas as alternativas, assim como a consulta anterior. O mesmo acontece na consulta da figura 4.31 abaixo.

```
1  -- UTILIZANDO INSTANTES
2  SELECT DISTINCT CHARACTER_NAME FROM CHARACTER WHERE STRENGTH > %LIMIT1% UNION ALL SELECT
   DISTINCT CHARACTER_NAME FROM CHARACTER WHERE STRENGTH < %LIMIT2% ORDER BY CHARACTER_NAME
3
4  -- UTILIZANDO PERÍODOS
5  SELECT DISTINCT CHARACTER_NAME FROM CHARACTER WHERE STRENGTH > %LIMIT1% UNION ALL SELECT
   DISTINCT CHARACTER_NAME FROM CHARACTER WHERE STRENGTH < %LIMIT2% ORDER BY CHARACTER_NAME
```

Figura 4.31: Implementação de mais uma consulta complexa envolvendo ambas as tabelas, em cada uma das alternativas estudadas

A consulta da figura 4.31 visa buscar todos os personagens que já tiveram força (campo "*Strength*") maior que *%LIMIT1%* ou menor que %*LIMIT2*%. Na consulta dessa figura, ambas as alternativas possuem implementação igual. As tabelas 14 e 15 mostram que essas duas consultas com condicionais mais complexos também tem execução parecida entre as alternativas e seguem a mesma tendência das consultas com condicionais mais simples, ou seja, um desempenho um pouco melhor da alternativa que utiliza períodos, mas com uma diferença pouco relevante entre elas.

Tabela 4.14: Tempo gasto na segunda consulta com condicionais complexos, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	3	1
2	2	2
3	3	0
4	4	1
5	3	0
6	3	0
Média	3	1

Tabela 4.15: Tempo gasto na terceira consulta com condicionais complexos, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	2369	2137
2	2151	1997
3	2121	2029
4	2111	2044
5	2231	2015
6	2149	2054
Média	2189	2046

Os resultados verificados até agora mostram que em consultas de seleção de dados, a alternativa que utiliza períodos leva vantagem. Entretanto, quando se muda o foco para as consultas que efetivamente alteram os dados, o comportamento apresentado é bem diferente.

Qualquer inserção ou atualização sobre as tabelas que representam entidades temporais requer uma checagem de chave primária temporal. Além disso, conforme apresentado na subseção anterior, existe a necessidade de se garantir a integridade das referências de chave estrangeira temporal entre essas tabelas em três situações: Inserções e atualizações de dados na tabela referenciante e atualizações e remoções na tabela referenciada. Pois são justamente esses tipos de consultas que são explorados a seguir. Na figura 4.32 são mostradas consultas de inserção sobre a tabela referenciante escritas para os fins dos experimentos realizados.

```
1  -- UTILIZANDO INSTANTES
2  INSERT INTO CHARACTER_CLASS VALUES (%CLASS_ID%,%CLASS_NAME%,7,-3,0,%CLASS_SINCE%);
3  INSERT INTO CHARACTER VALUES (%CHAR_ID%,%CHAR_NAME%,312,0,1,61,22,41, %CHAR_SINCE% ,
   %CLASS_ID%);
4
5  -- UTILIZANDO PERÍODOS
6  INSERT INTO CHARACTER_CLASS VALUES (%CLASS_ID%,%CLASS_NAME%,7,-3,0,%CLASS_FROM% ,
   %CLASS_TO%);
7  INSERT INTO CHARACTER VALUES (%CHAR_ID%,%CHAR_NAME%,312,0,1,61,22,41,%CHAR_FROM%,%CHAR_TO%
    ,%CLASS_ID%);
```

Figura 4.32: Implementação de consultas de inserção de dados na tabela referenciante, em cada uma das alternativas estudadas

As consultas de inserção são bastante simples em cada uma das alternativas. Varia apenas o fato de que a alternativa que utiliza períodos requer que seja informado o final da validade daquele registro na inserção. No caso das consultas mostradas na figura, usam-se marcadores para identificar esses pontos no tempo, %*SINCE*%, %*FROM*% e %*TO*%. Além disso, para que a inserção na tabela referenciante funcione, é necessário que os valores da entidade referenciada pelo valor %*class_id*% na tabela referenciada "cubram" a validade do registro inserido na tabela referenciante, o que é justamente o papel que os procedimentos de verificação de integridade das referências executam. Por esse motivo, é inserido um registro na tabela referenciada antes da tabela referenciante, garantindo o sucesso da inserção na tabela referenciante.

Ao serem executadas, essas consultas requerem as seguintes verificações: chave primária temporal, uma vez para cada inserção, e chave estrangeira temporal, quando a inserção na tabela referenciante ocorre. A chave primária temporal, na alternativa que utiliza instantes, é garantida pelo SGBD, já que se usa uma chave primária simples para

a sua implementação. Os outros procedimentos, em ambas as alternativas, são disparados por *triggers*, cujos resultados de execução indicam se a inserção irá funcionar ou falhar. Na Tabela 4.16 abaixo, é possível verificar em números o impacto da necessidade de execução desses procedimentos sobre cada uma das alternativas.

Tabela 4.16: Tempo gasto para inserções sobre as tabelas, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	2	43
2	4	32
3	2	16
4	2	16
5	3	32
6	2	15
Média	3	26

A Tabela 4.16 acima mostra que a execução das inserções na alternativa utilizando períodos teve um desempenho muito pior. Naturalmente os números são baixos por conta do fato de que são apenas duas instruções simples. Mas é interessante notar que em média o desempenho da alternativa utilizando períodos foi quase nove vezes pior que o da alternativa que utiliza instantes. No caso de inserções em *batch*, essa diferença acaba sendo muito mais notável, conforme verificado na Tabela 4.17 abaixo.

Tabela 4.17: Tempo gasto para inserções em *batch* sobre as tabelas, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	81614	1024043
2	80506	1049525
3	80990	1023561
4	86597	918197
5	86692	919886
6	92605	936999
Média	84834	978702

A Tabela 4.17 mostra uma tendência já indicada na Tabela 4.16: as inserções sobre as tabelas da alternativa que utiliza períodos são muito mais custosas, sendo que quando as operações são em batch a diferença entre o tempo médio de execução do mesmo número de inserções para ambas as alternativas pulou de nove para mais de onze vezes maior na alternativa que utiliza períodos. Essa diferença se dá pelo tempo gasto na execução dos procedimentos que verificam as restrições de integridade de chaves primárias temporais, em ambas as tabelas, e de chaves estrangeiras temporais, na tabela referenciada, usando como base o valor da tabela referenciante.

Além de inserções, as atualizações também estão sujeitas ao efeito de procedimentos de verificação. Atualizações e remoções sobre a tabela referenciada necessitam que seja verificado que não existe uma violação das referências de chave estrangeira temporal provenientes da tabela referenciante. Já atualizações sobre a tabela referenciante estão sujeitas às mesmas verificações feitas em inserções, já exploradas anteriormente. Por

esse motivo, o foco dos experimentos a seguir foi sobre atualizações e remoções na tabela referenciada. As figuras 4.33 e 4.34 abaixo mostram como são escritas consultas desse tipo em ambas as alternativas.

```
-- UTILIZANDO INSTANTES
UPDATE CHARACTER_CLASS SET STR_MODIFIER = %STR_VALUE% WHERE CLASS_ID = %CLASS_ID% AND
VALID_SINCE = %TARGET_INSTANT%;

-- UTILIZANDO PERÍODOS
UPDATE CHARACTER_CLASS SET STR_MODIFIER = %STR_VALUE% WHERE CLASS_ID = %CLASS_ID% AND
VALID_FROM = %TARGET_INSTANT%
```

Figura 4.33: Implementação de consultas de atualização de dados na tabela referenciada, em cada uma das alternativas estudadas

A consulta da figura 4.33 mostra a atualização do valor *STR_MODIFIER* na tabela referenciada usando como parâmetros um id de classe (%*CLASS_ID*%) e um instante específico (%*TARGET_INSTANT*%) em ambas as alternativas. Como é possível perceber ambas são consultas bem simples. Entretanto, mesmo assim, se mostram bastante pesadas para o sistema, como mostrado na Tabela 4.18 abaixo.

Tabela 4.18: Tempo gasto para atualizações sobre as tabelas, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	590	23509
2	608	23821
3	598	23697
4	586	23649
5	593	23821
6	588	25118
Média	594	23936

A Tabela 4.18 acima mostra que uma simples atualização leva muito mais tempo para ser concluída que uma seleção ou inserção, em qualquer uma das alternativas. Além disso, existe uma diferença enorme de desempenho entre as alternativas, com larga vantagem para a alternativa que utiliza instantes. Isso se deve à execução dos procedimentos para verificação de integridade de chaves temporais. Cada registro referente à entidades que possuem referências para a entidade que está sendo alterada precisa ser verificado, para garantir que a alteração não cria uma falha temporal na referência de chave estrangeira. Como foi verificado no capítulo anterior, o procedimento é muito mais simples na alternativa que utiliza instantes, e isso se reflete diretamente na performance de cada consulta. Em relação a remoções, o comportamento observado foi parecido. A figura 4.34 mostra como são escritas consultas simples de remoções em cada uma das alternativas.

```
-- UTILIZANDO INSTANTES
DELETE FROM CHARACTER_CLASS WHERE CLASS_ID = %CLASS_ID% AND VALID_SINCE = %TARGET_INSTANT%;

-- UTILIZANDO PERÍODOS
DELETE FROM CHARACTER CLASS WHERE CLASS ID = %CLASS ID% AND VALID FROM = %TARGET INSTANT%;
```

Figura 4.34: Implementação de consultas de remoção de dados na tabela referenciada, em cada uma das alternativas estudadas

As consultas da figura 4.34 mostram a remoção de um registro cuja validade se inicia no instante (%*TARGET_INSTANT*%) e que possui o id de classe identificado por %*CLASS_ID*%. Assim como atualizações, são consultas bastante simples. E assim como as atualizações, se verificou serem extremamente pesadas para o sistema. Na Tabela 4.19 abaixo é possível verificar o resultado da execução dessas consultas.

Tabela 4.19: Tempo gasto para remoções sobre as tabelas, em milissegundos

Run id/Alternativa	Utilizando Instantes	Utilizando Períodos
1	293	23509
2	299	23665
3	298	23495
4	299	23806
5	299	23727
6	309	24055
Média	300	23710

A Tabela 4.19 acima mostra que o desempenho para consultas de remoção foi muito parecido com o desempenho para consultas de atualização, apresentando uma melhora considerável na alternativa que utiliza instantes e uma melhora discreta na alternativa que utiliza períodos. Isso se deve ao fato de que remoções precisam fazer a mesma verificação que atualizações no que diz respeito a referências de chave estrangeira temporal, mas não precisam fazer a verificação de chave primária temporal que as atualizações também precisam fazer, por isso o desempenho superior de consultas de remoção.

Analisando os resultados é possível verificar que ambas as alternativas possuem vantagens e desvantagens, devendo a escolha por uma delas ser guiada pelo contexto da aplicação. A alternativa que utiliza instantes tem desempenho pior em consultas, especialmente nas que envolvem muitas operações e condicionais temporais. Suas consultas também são ligeiramente mais complexas de serem escritas. Entretanto, essa alternativa é mais simples de se implementar porque requer menos procedimentos de verificação de integridade temporal, podendo usar construtores nativos do SGBD pra isso. Já a alternativa que utiliza períodos depende que existam procedimentos complexos de verificação de integridade, o que causa impacto considerável no desempenho de comandos para alteração dos dados da base. Por isso, o modelo escolhido deve levar em consideração qual característica da aplicação é necessário favorecer. Em um contexto onde ocorrem atualizações de dados frequentemente, a alternativa que utiliza instantes é a mais indicada. Já em um contexto onde consultas acontecem com frequência maior que atualizações, é mais indicado que se utilize a alternativa que utiliza períodos.

4.4.3 Alternativas para implementação de um registro de operações

No capítulo anterior foram estudadas quatro alternativas para implementação de um registro de operações sobre uma determinada tabela em uma base de dados. Para facilitar a escolha do projetista da base de dados por uma dessas opções ao projetar uma estrutura desse tipo, foram conduzidos experimentos sobre essas alternativas, que serão detalhados na sequência dessa seção.

Para esses experimentos, será utilizado o mesmo contexto utilizado no capítulo anterior para exemplificar o uso de cada alternativa. De modo geral, a tabela de monitoramento e a tabela monitorada são da forma mostrada na figura 4.35, onde são expostas as suas descrições no modelo relacional.

Character (**Character_id**, Character_name, Speed, Strength, Inteligence, Char_Class_Name)

C_Log (**Character_id**, Character_name, Speed, Strength, Inteligence, Char_Class_Name, When_Changed)

Figura 4.35: Tabelas monitorada e de monitoramento no modelo relacional

Conforme pode ser verificado na figura 4.35, as tabelas são praticamente iguais, tendo a tabela de monitoramento um campo a mais, referente ao instante relacionado à geração de cada tupla que existe nela. Como visto no capitulo anterior, duas alternativas de implementação utilizam também um campo que identifica qual a operação que gerou a tupla na tabela de monitoramento, estando esse campo também presente na tabela de monitoramento.

Cada uma das quatro alternativas estudadas no capitulo anterior apresenta limitações e facilitadores. A alternativa sem inserções, como o nome diz, é a alternativa onde operações de inserção não são registrados na tabela de monitoramento. Essa alternativa é a mais básica de todas e serve de base para a implementação das outras. Já a alternativa com inserções é apenas a alternativa sem inserções acrescida de um procedimento que gera dados na tabela monitorada também para inserções. A alternativa com registro de operações adiciona um campo na tabela de monitoramento, referente a operação que gerou o registro, utilizado para possibilitar maior precisão na reconstrução da tabela monitorada em um ponto no tempo. Essas três alternativas utilizam imagens anteriores ao armazenar dados na tabela de monitoramento. Assim, uma última alternativa que utiliza imagens posteriores também foi implementada e testada.

4.4.3.1 Preparação

Para que os experimentos fossem rodados sobre um número relevante de dados, as tabelas foram populadas da seguinte maneira: primeiramente, foram gerados comandos de inserção de cem mil registros. Após esses registros serem executados, todos os registros sofrem atualizações em *batch*, de mil em mil, gerando registros na tabela de monitoramento. Por fim, metade dos cem mil registros gerados são removidos, restando apenas cinquenta mil registros na tabela monitorada e gerando mais cinquenta mil registros na tabela de monitoramento. Esse mesmo procedimento é aplicado para todas as alternativas estudadas. Naturalmente, dependendo da característica de cada alternativa um volume maior ou menor de dados é gerado.

4.4.3.2 Análise sobre o Volume de Dados

As operações utilizadas para popular a tabela monitorada geram um volume de dados diferente sobre cada uma das alternativas. A primeira diferença ocorre no número de registros. Por não registrar inserções na tabela de monitoramento, a alternativa sem inserções tem cem mil registros a menos que as outras, o que naturalmente se reflete no volume de dados apresentado. As outras alternativas, mesmo tendo um número igual de registros, ainda assim apresentam diferenças no volume de dados, conforme mostrado na Tabela 4.20 abaixo. Nessa tabela, as alternativas são rotuladas segundo a seguinte legenda: SI para a alternativa sem inserções, CI para a alternativa com inserções, COIA para a alternativa com registro de alterações, utilizando imagens anteriores, e COIP para

a alternativa com registro de operações utilizando imagens posteriores. Essa legenda é seguida em todas as tabelas referentes a essas alternativas.

Tabela 4.20: Volume de dados de cada uma das alternativas

Alternativa	SI	CI	COIA	COIP
Volume de Dados	40MB	72MB	88 MB	106 MB

Na Tabela 4.20 acima é possível verificar em números como as regras de população da tabela de monitoramento de cada alternativa afetam cada uma delas. O baixo volume de dados da alternativa sem inserções é explicado pelos muitos registros a menos provenientes das inserções não armazenadas na tabela de monitoramento. A alternativa com inserções, que possui esses registros, tem um aumento proporcional ao número de registros a mais que possui. Já a alternativa com registro de operações apresenta um volume um pouco superior, devido ao fato de conter um campo a mais do que a alternativa anterior. Por fim, a alternativa com imagens posteriores apresenta um volume maior que a anterior porque, ao utilizar imagens posteriores na tabela de monitoramento, a mesma acaba sendo menos esparsa que na alternativa anterior.

No caso das três primeiras alternativas, os valores a serem inseridos na tabela de monitoramento são os valores que os registros possuíam. No caso de inserções, todos os valores a não ser o identificador de cada registro são nulos. Esse é comportamento ao se utilizar imagens anteriores de cada registro na tabela de monitoramento. Já ao utilizar imagens posteriores, as inserções são registradas com os valores que vão ser inseridos, ao passo que as remoções de dados são inseridas com valores nulos. Como mostrado na subseção anterior, cem mil registros foram inseridos mas apenas metade deles foram removidos posteriormente. Isso explica porque a tabela de monitoramento, na alternativa com registro de operações utilizando imagens anteriores é mais esparsa que a mesma tabela na alternativa que utiliza imagens posteriores. As cem mil inserções são registradas com todos os valores na alternativa que utiliza imagens posteriores, enquanto as cinquenta mil remoções são registradas com quase todos os campos com valores nulos. Na alternativa com que utiliza imagens anteriores, o comportamento é o inverso. Como foram feitas mais inserções que remoções, o volume de dados na alternativa com imagens posteriores acaba sendo o maior de todos, já que valores nulos requerem menos espaço do que valores concretos para serem armazenados (MYSQL, 2012).

A partir dessas observações, podem-se tirar algumas conclusões. Naturalmente que a alternativa que não registra inserções é a mais econômica em termos de espaço, mas dificilmente ela pode ser aplicada num contexto real. Entre as outras três alternativas, é possível verificar que a adição de um campo para registrar a operação sendo efetuada causa um aumento pequeno no volume de dados (esse valor ficou em 16% nos experimentos realizados). Já a diferença verificada entre as duas alternativas que possuem esse campo extra ilustra que utilizar imagens posteriores ou anteriores afetará o volume de dados de acordo com o tipo de alteração sobre os dados que for mais frequente. Em um contexto em que inserções são mais comuns que remoções, utilizar imagens anteriores resultará em economia de espaço em disco. Nos experimentos realizados, essa economia foi de aproximadamente 21%.

4.4.3.3Análise sobre a necessidade de verificações de integridade externas

No capítulo anterior, na seção onde cada alternativa de implementação de um registro de operações é definida, já foram apresentadas as funções e *triggers* responsáveis por popular a tabela de monitoramento. Basicamente, todas as funções mostradas inserem, a cada alteração na tabela monitorada, um registro na tabela de monitoramento correspondente ao registro da tabela monitorada que está sofrendo a alteração. A simplicidade de todos os procedimentos é muito parecida, variando muito pouco em função da necessidade de identificação da operação realizada no campo correspondente e do uso de imagens posteriores ou anteriores. Entretanto, mesmo essas diferenças são muito pequenas e pouco influenciam na execução de cada procedimento, sendo o grande diferencial em termos de desempenho das alternativas a simplicidade de escrita de cada consulta, analisada na seção seguinte.

4.4.3.4Execução e Análise de Simplicidade de Consultas

Para possibilitar uma análise do desempenho de cada uma das alternativas apresentadas, foi elaborado um conjunto de consultas, cada consulta ambicionando avaliar uma situação diferente, passível de acontecer em um contexto real.

As consultas elaboradas para esses experimentos foram: consultas para reconstrução da tabela monitorada num ponto específico no tempo, consultas para tradução da tabela monitorada em uma tabela com períodos de validade, atualizações simples, remoções simples, e consultas diretamente sobre a tabela de monitoramento.

Um dos principais possíveis usos da tabela de monitoramento é utilizá-la para reconstruir a tabela monitorada em um ponto no tempo. Isso poderia ser utilizado tanto para verificar o estado da tabela em um determinado instante quanto para restaurar a tabela para um estado anterior no caso de alterações indevidas terem sido feitas sobre ela. Na figura 4.36 abaixo é possível verificar como ficam consultas desse tipo em cada uma das alternativas.

```
-- SEM INSERÇÕES
SELECT * FROM CHARACTER AS P WHERE NOT EXISTS (SELECT * FROM C_LOG AS A WHERE P.
CHARACTER_ID = A.CHARACTER_ID AND          A.WHEN_CHANGED > %TARGET_INSTANT%) UNION
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, CHARACTER_LEVEL, INTELIGENCE ,
CHARACTER_CLASS FROM C_LOG AS A WHERE WHEN_CHANGED = (SELECT MIN(WHEN_CHANGED) FROM C_LOG
 AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A2.WHEN_CHANGED > %TARGET_INSTANT%)

-- COM INSERÇÕES
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, CHARACTER_LEVEL, INTELIGENCE,
CHARACTER_CLASS FROM C_LOG AS A WHERE NOT EXISTS ( SELECT * FROM C_LOG AS A2 WHERE A.
CHARACTER_ID = A2.CHARACTER_ID AND %TARGET_INSTANT% < A2.WHEN_CHANGED AND A2.WHEN_CHANGED
 < A.WHEN_CHANGED) AND %TARGET_INSTANT% < A.WHEN_CHANGED AND EXISTS ( SELECT * FROM C_LOG
 AS A3 WHERE A.CHARACTER_ID = A3.CHARACTER_ID AND A3.WHEN_CHANGED <= %TARGET_INSTANT%)
UNION SELECT * FROM CHARACTER AS P WHERE %TARGET_INSTANT% > (SELECT MAX(WHEN_CHANGED)
FROM C_LOG AS A WHERE P.CHARACTER_ID = A.CHARACTER_ID)

-- COM REGISTRO DE OPERAÇÕES
SELECT * FROM CHARACTER AS P WHERE %TARGET_INSTANT% > (SELECT MAX(WHEN_CHANGED) FROM
C_LOG AS A WHERE P.CHARACTER_ID = A.CHARACTER_ID) UNION SELECT A2.CHARACTER_ID, A2.
CHARACTER_NAME, A2.SPEED, A2.STRENGTH, A2.INTELIGENCE, A2.CHARACTER_LEVEL, A2.
CHARACTER_CLASS FROM C_LOG AS A, C_LOG AS A2 WHERE A.WHEN_CHANGED < A2.WHEN_CHANGED AND A
.CHARACTER_ID = A2.CHARACTER_ID AND NOT EXISTS (SELECT * FROM C_LOG AS A3 WHERE A.
CHARACTER_ID = A3.CHARACTER_ID AND A.WHEN_CHANGED < A3.WHEN_CHANGED AND A3.WHEN_CHANGED <
 A2.WHEN_CHANGED) AND A.WHEN_CHANGED < %TARGET_INSTANT% AND %TARGET_INSTANT% < A2.
WHEN_CHANGED AND A.OPERATION <> 'DELETE'

-- COM IMAGENS POSTERIORES
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
CHARACTER_CLASS FROM C_LOG AS A WHERE          A.WHEN_CHANGED =(SELECT MAX(A2.WHEN_CHANGED
) FROM C_LOG AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A2.WHEN_CHANGED < %
TARGET_INSTANT%) AND A.OPERATION <> 'DELETE'
```

Figura 4.36: Algoritmo de reconstrução da tabela monitorada em um determinado instante no tempo

No capítulo anterior, as consultas da figura 4.36 já haviam sido exploradas do ponto de vista da simplicidade de cada uma e limitações que cada uma apresenta. Em uma breve recapitulação, as características dessas consultas, de cada alternativa, são as seguintes: na alternativa sem inserções, não se sabe quando os dados foram inseridos, e por isso, qualquer que seja o instante alvo da reconstrução (%*TARGET_INSTANT*% na figura) todos os dados que foram inseridos na tabela irão constar no resultado final, mesmo que tenham sido inseridos depois de %*TARGET_INSTANT*%. Em um modelo onde as inserções são recordadas, é impossível distinguir uma deleção seguida de uma inserção da mesma entidade na tabela monitorada de duas atualizações subsequentes, o que pode levar a erros ao reconstruir a tabela. Nas alternativas onde há registro de operações, não há limitações, mas os algoritmos são mais complexos, sendo os da alternativa que utiliza imagens anteriores mais complexos que o da alternativa que utiliza imagens posteriores. As implicações dessas caracteristicas, em termos de desempenho, podem ser visualizadas na tabela 4.21.

Tabela 4.21: Desempenho de consultas de reconstrução da tabela monitorada, com base na tabela de monitoramento para cada uma das alternativas

Run ID/Alternativa	SI	CI	COIA	COIP
1	3044	14696	6613	3181
2	2879	10425	7732	4039
3	2703	10878	5141	3613
4	2881	11173	9935	3034
5	2709	10841	5939	2964
6	2820	10022	6801	4099
Média	2839	11339	7027	3488

Na Tabela 4.21 é possível verificar que o desempenho da alternativa sem inserções é o melhor de todos. Isso se deve ao fato de que a consulta também é a mais simples se comparada às outras. Nela, se verifica qual era o estado de cada entidade presente na tabela monitorada, de acordo com os registros da tabela de monitoramento e se adicionam ao resultado final registros da tabela monitorada cujas entidades não existam na tabela de monitoramento, ou seja, não foram alterados desde a sua inserção.

Já na alternativa em que as inserções são registradas, diminui a simplicidade da consulta. Agora ela precisa considerar o momento em que os registros foram inseridos sobre a tabela, além da junção com a tabela monitorada para o caso do registro da mesma ser o atual para o instante desejado. Essa redução de simplicidade aparece nos números da Tabela 4.21, com essa alternativa apresentando um desempenho quatro vezes pior que a alternativa que não grava inserções. Além disso, essa alternativa possui a limitação referente a não poder diferenciar remoções e reinserções de uma entidade de duas atualizações subsequentes, o que é resolvido ao se armazenar a operação que gera cada registro da tabela de monitoramento.

A alternativa que possui esse registro de operações, utilizando imagens anteriores, também apresenta uma consulta complexa, uma vez que agora é necessário se levar em consideração o campo de operação ao montar o resultado final da consulta. Entretanto, a existência desse campo pode acabar auxiliando o desempenho da consulta uma vez que registros marcados como sendo de operações de remoção são mais facilmente eliminados. Essa característica aparece nos resultados da Tabela 4.21, onde o desempenho verificado para consultas de reconstrução, para essa alternativa, é superior ao da alternativa anterior.

Por fim, a alternativa que registra operações, mas armazena imagens posteriores dos registros na tabela monitorada apresenta uma consulta relativamente simples se comparada às outras. Ela é composta de apenas uma consulta que possui uma subconsulta em um de seus condicionais, o outro sendo referente à operação registrada. Essa simplicidade se traduz em eficiência na execução da consulta, sendo aproximadamente 50% mais eficiente que a outra alternativa com registro de operações, e tendo desempenho comparável ao da alternativa que não registra inserções, mas com o grande ganho de não apresentar nenhuma limitação ao reconstruir a tabela monitorada.

Já nas consultas para tradução das tabelas para uma tabela com períodos, os tempos de execução verificados foram bastante similares entre as alternativas. O que se pode averiguar na figura 4.37 abaixo são as diferenças na simplicidade de escrita das consultas, e nas restrições de cada uma delas, já discutidas no capítulo anterior.

```
-- SEM INSERÇÕES
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED,STRENGTH, CHARACTER_LEVEL, INTELIGENCE,%
STARTING_INSTANT%, CURRENT_TIMESTAMP FROM CHARACTER WHERE NOT EXISTS ( SELECT * FROM
C_LOG WHERE C_LOG.CHARACTER_ID = CHARACTER.CHARACTER_ID) UNION SELECT CHARACTER_ID,
CHARACTER_NAME, SPEED, STRENGTH, CHARACTER_LEVEL, INTELIGENCE,%STARTING_INSTANT%,
WHEN_CHANGED FROM C_LOG AS A1 WHERE NOT EXISTS ( SELECT * FROM C_LOG AS A2 WHERE A1.
CHARACTER_ID = A2.CHARACTER_ID AND A1.WHEN_CHANGED > A2.WHEN_CHANGED) UNION SELECT A1.
CHARACTER_ID, A1.CHARACTER_NAME, A1.SPEED,A1.STRENGTH, A1.CHARACTER_LEVEL, A1.INTELIGENCE
,WHEN_CHANGED, CURRENT_TIMESTAMP FROM C_LOG AS A1, CHARACTER WHERE A1.CHARACTER_ID =
CHARACTER.CHARACTER_ID AND NOT EXISTS ( SELECT * FROM C_LOG AS A2 WHERE A1.CHARACTER_ID =
 A2.CHARACTER_ID AND A1.WHEN_CHANGED < A2.WHEN_CHANGED) UNION SELECT A1.CHARACTER_ID, A1.
CHARACTER_NAME, A1.SPEED, A1.STRENGTH, A1.CHARACTER_LEVEL, A1.INTELIGENCE, A0.
WHEN_CHANGED, A1.WHEN_CHANGED FROM C_LOG AS A0, C_LOG AS A1 WHERE A0.CHARACTER_ID = A1.
CHARACTER_ID AND A0.WHEN_CHANGED < A1.WHEN_CHANGED AND NOT EXISTS ( SELECT * FROM C_LOG
AS M WHERE M.CHARACTER_ID = A1.CHARACTER_ID AND M.WHEN_CHANGED < A1.WHEN_CHANGED AND M.
WHEN_CHANGED > A0.WHEN_CHANGED)

-- COM INSERÇÕES
SELECT A3.CHARACTER_ID, A3.CHARACTER_NAME, A3.SPEED, A3.STRENGTH, A3.CHARACTER_LEVEL,
A3.INTELIGENCE, A3.CHARACTER_CLASS, WHEN_CHANGED, CURRENT_TIMESTAMP FROM C_LOG
AS A1, CHARACTER A3 WHERE A1.CHARACTER_ID = A3.CHARACTER_ID AND NOT EXISTS ( SELECT *
FROM C_LOG AS A2 WHERE A1.CHARACTER_ID = A2.CHARACTER_ID AND A1.WHEN_CHANGED < A2.
WHEN_CHANGED) UNION SELECT A1.CHARACTER_ID, A1.CHARACTER_NAME, A1.SPEED, A1.STRENGTH, A1.
CHARACTER_LEVEL, A1.INTELIGENCE, A1.CHARACTER_CLASS, A0.WHEN_CHANGED, A1.WHEN_CHANGED
FROM C_LOG AS A0, C_LOG AS A1 WHERE A0.CHARACTER_ID = A1.CHARACTER_ID AND A0.WHEN_CHANGED
 < A1.WHEN_CHANGED AND NOT EXISTS (SELECT * FROM C_LOG AS M WHERE M.CHARACTER_ID = A1.
CHARACTER_ID AND M.WHEN_CHANGED < A1.WHEN_CHANGED AND M.WHEN_CHANGED > A0.WHEN_CHANGED)

-- COM REGISTRO DE OPERAÇÕES
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
CHARACTER_CLASS, WHEN_CHANGED,TIMEFIELD FROM ( SELECT A1.CHARACTER_ID, A1.CHARACTER_NAME,
 A1.SPEED, A1.STRENGTH, A1.INTELIGENCE, A1.CHARACTER_LEVEL,CHARACTER_CLASS, WHEN_CHANGED,
 CURRENT_TIMESTAMP AS TIMEFIELD, A1.OPERATION FROM C_LOG AS A1, CHARACTER WHERE A1.
CHARACTER_ID = CHARACTER.CHARACTER_ID AND NOT EXISTS ( SELECT * FROM C_LOG AS A2 WHERE A1
.CHARACTER_ID = A2.CHARACTER_ID AND A1.WHEN_CHANGED < A2.WHEN_CHANGED) ) AS FOO1 UNION
SELECT CHARACTER_ID, CHARACTER_NAME, SPEED,STRENGTH, INTELIGENCE, CHARACTER_LEVEL,
CHARACTER_CLASS, START_DATE,END_DATE FROM (SELECT A1.CHARACTER_ID, A1.CHARACTER_NAME, A1.
SPEED,A1.STRENGTH, A1.INTELIGENCE, A1.CHARACTER_LEVEL, A0.CHARACTER_CLASS ,A0.
WHEN_CHANGED AS START_DATE, A1.WHEN_CHANGED AS END_DATE, A0.OPERATION FROM C_LOG AS A0,
C_LOG AS A1 WHERE A0.CHARACTER_ID = A1.CHARACTER_ID AND A0.WHEN_CHANGED < A1.WHEN_CHANGED
 AND NOT EXISTS ( SELECT * FROM C_LOG AS M WHERE M.CHARACTER_ID = A1.CHARACTER_ID AND M.
WHEN_CHANGED < A1.WHEN_CHANGED AND M.WHEN_CHANGED > A0.WHEN_CHANGED)) AS FOO1 WHERE FOO1.
OPERATION <> 'DELETE'

-- COM IMAGENS POSTERIORES
SELECT A.CHARACTER_ID, A.CHARACTER_NAME, A.SPEED, A.STRENGTH, A.INTELIGENCE, A.
CHARACTER_LEVEL,CHARACTER_CLASS, A.WHEN_CHANGED, A2.WHEN_CHANGED FROM C_LOG AS A, C_LOG
AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A.WHEN_CHANGED < A2.WHEN_CHANGED AND A.
OPERATION <> 'DELETE' AND NOT EXISTS ( SELECT * FROM C_LOG AS A3 WHERE A.CHARACTER_ID =
A3.CHARACTER_ID AND A.WHEN_CHANGED < A3.WHEN_CHANGED AND A3.WHEN_CHANGED < A2.
WHEN_CHANGED) UNION SELECT CHARACTER_ID, CHARACTER_NAME, SPEED, STRENGTH, INTELIGENCE,
CHARACTER_LEVEL, WHEN_CHANGED, CURRENT_DATE FROM C_LOG AS A WHERE A.OPERATION <> 'DELETE'
 AND NOT EXISTS ( SELECT * FROM C_LOG AS A3 WHERE A.CHARACTER_ID = A3.CHARACTER_ID AND A.
WHEN_CHANGED < A3.WHEN_CHANGED)
```

Figura 4.37: Algoritmo de tradução da tabela de monitoramento em uma tabela de períodos

Nas consultas da figura 4.37 é possível perceber que as consultas das alternativas sem inserções e com registro de operações, utilizando imagens anteriores, são as mais complexas. Na consulta da alternativa sem inserções, dados que existem na tabela monitorada e não existem na tabela de monitoramento são adicionados no resultado final com um instante inicial pré-definido na consulta (%*STARTING_INSTANT*% na figura). Além disso, é necessário buscar na tabela monitorada também os valores dos registros que atualmente são válidos. Os registros intermediários e suas validades são retirados da tabela de monitoramento. Na alternativa com inserções, isso não é necessário porque as inserções estão disponíveis na tabela, só sendo necessário consultar a tabela monitorada para pegar os valores atuais das entidades.

A consulta da alternativa com registro de operações, mas com imagens anteriores, é bastante parecida com a consulta da alternativa com inserções. A diferença principal é que é preciso associar um *alias* ao resultado da consulta original, para testar e excluir registros associados a operações de remoção do resultado final. Já a consulta da alternativa com registro de operações, mas utilizando imagens posteriores, é muito mais simples pois não precisa de uma junção com a tabela monitorada, todos os dados já estão disponíveis na tabela de monitoramento. As variações de performance entre as alternativas podem ser vistas na Tabela 4.22 abaixo.

Tabela 4.22: Desempenho de consultas de tradução da tabela monitorada e da tabela de monitoramento em uma tabela com períodos de validade

Run ID/Alternativa	SI	CI	COIA	COIP
1	7022	7793	11174	8259
2	6148	9657	10030	9835
3	8300	8874	12653	8373
4	9785	9748	10002	7836
5	7720	8394	8997	7819
6	7757	9784	9236	9462
Média	7789	9042	10349	8597

A alternativa sem inserções, mesmo sendo uma das mais complexas, tem muitos valores pré-fixados no seu resultado, não necessitando extraí-los das tabelas, por conta da falta das inserções na tabela de monitoramento. Isso ajuda a melhorar o desempenho de suas consultas. Já a alternativa com inserções e a alternativa com registro de operações, utilizando imagens posteriores tem desempenho bem parecido. A alternativa com inserções não necessita lidar com o campo adicional referente à operação, enquanto a alternativa com registro de operações utilizando imagens posteriores se beneficia de poder buscar todos os dados na tabela de monitoramento. Essas características se refletem nos valores encontrados nos experimentos.

Da mesma maneira, consultas de atualizações e remoções de dados simples sobre a tabela monitorada também apresentaram pequena variação de performance. Como é possível ver nas figuras 4.38 e 4.39, as consultas, do ponto de vista de sua escrita, não sofrem nenhuma alteração por conta da existência de um registro de alterações e são as mesmas para todas as alternativas.

```
1  UPDATE CHARACTER SET SPEED = %TARGET_SPEED_VALUE% WHERE CHARACTER_ID > %UPDATE_LIMIT1% AND
    CHARACTER_ID < %UPDATE_LIMIT2%
```

Figura 4.38: Atualização simples sobre a tabela monitorada

```
1  DELETE FROM CHARACTER WHERE CHARACTER_ID > %DELETE_LIMIT1% AND CHARACTER_ID < %
   DELETE_LIMIT2%
```

Figura 4.39: Remoção simples sobre a tabela monitorada

A diferença, nesse caso, está nos procedimentos auxiliares que são executados quando essas operações são executadas. Mas mesmo esses, como mostrado na subseção anterior, são extremamente simples e muito parecidos. Dessa maneira, é natural que não haja praticamente nenhuma diferença de performance entre as alternativas, como mostrado nas Tabela 4.23 e 24 abaixo.

Tabela 4.23: Desempenho de consultas de atualização sobre a tabela monitorada

Run ID/Alternativa	SI	CI	COIA	COIP
1	16	15	20	15
2	15	15	32	14
3	16	32	20	15
4	15	16	23	16
5	31	30	21	16
6	16	16	30	16
Média	18	21	24	15

Tabela 4.24: Desempenho de consultas de remoção sobre a tabela monitorada

Run ID/Alternativa	SI	CI	COIA	COIP
1	14	20	15	14
2	16	14	16	15
3	15	16	15	16
4	14	20	14	16
5	15	16	22	15
6	16	20	21	15
Média	15	18	17	15

Por fim, foram analisadas consultas de seleção temporais diretamente sobre o registro de operações. Nessas consultas, se verificou um padrão de desempenho semelhante ao encontrado nas outras consultas. Na figura 4.40 é possível ver como fica um exemplo de seleção temporal diretamente sobre a tabela monitorada, em cada uma das alternativas.

```
1  -- SEM INSERÇÕES
2  SELECT CHARACTER_LEVEL FROM CHARACTER AS P WHERE NOT EXISTS (SELECT * FROM C_LOG AS A
   WHERE P.CHARACTER_ID = A.CHARACTER_ID AND          A.WHEN_CHANGED > %TARGET_INSTANT%)
   UNION SELECT CHARACTER_LEVEL FROM C_LOG AS A WHERE WHEN_CHANGED = (SELECT MIN(
   WHEN_CHANGED) FROM C_LOG AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A2.WHEN_CHANGED
    > %TARGET_INSTANT%)
3
4  -- COM INSERÇÕES
5  SELECT CHARACTER_LEVEL FROM C_LOG AS A WHERE NOT EXISTS ( SELECT * FROM C_LOG AS A2 WHERE
    A.CHARACTER_ID = A2.CHARACTER_ID AND %TARGET_INSTANT% < A2.WHEN_CHANGED AND A2.
   WHEN_CHANGED < A.WHEN_CHANGED) AND %TARGET_INSTANT% < A.WHEN_CHANGED AND EXISTS ( SELECT
   * FROM C_LOG AS A3 WHERE A.CHARACTER_ID = A3.CHARACTER_ID AND A3.WHEN_CHANGED <= %
   TARGET_INSTANT%) UNION SELECT CHARACTER_LEVEL FROM CHARACTER AS P WHERE %TARGET_INSTANT%
   > (SELECT MAX(WHEN_CHANGED) FROM C_LOG AS A WHERE P.CHARACTER_ID = A.CHARACTER_ID)
6
7  -- COM REGISTRO DE OPERAÇÕES
8  SELECT CHARACTER_LEVEL FROM CHARACTER AS P WHERE %TARGET_INSTANT% > (SELECT MAX(
   WHEN_CHANGED) FROM C_LOG AS A WHERE P.CHARACTER_ID = A.CHARACTER_ID) UNION SELECT A2.
   CHARACTER_LEVEL FROM C_LOG AS A, C_LOG AS A2 WHERE A.WHEN_CHANGED < A2.WHEN_CHANGED AND A
   .CHARACTER_ID = A2.CHARACTER_ID AND NOT EXISTS (SELECT * FROM C_LOG AS A3 WHERE A.
   CHARACTER_ID = A3.CHARACTER_ID AND A.WHEN_CHANGED < A3.WHEN_CHANGED AND A3.WHEN_CHANGED <
    A2.WHEN_CHANGED) AND A.WHEN_CHANGED < %TARGET_INSTANT% AND %TARGET_INSTANT% < A2.
   WHEN_CHANGED AND A.OPERATION <> 'DELETE'
9
10 -- COM REGISTRO DE OPERAÇÕES E IMAGENS POSTERIORES
11 SELECT CHARACTER_LEVEL FROM C_LOG AS A WHERE          A.WHEN_CHANGED =(SELECT MAX(A2.
   WHEN_CHANGED) FROM C_LOG AS A2 WHERE A.CHARACTER_ID = A2.CHARACTER_ID AND A2.WHEN_CHANGED
    < %TARGET_INSTANT%) AND A.OPERATION <> 'DELETE'
```

Figura 4.40: Seleção temporal diretamente sobre a tabela de monitoramento

Todas as consultas da figura 4.40 buscam recuperar o valor do campo *CHARACTER_LEVEL* no instante *%TARGET_INSTANT%*. Nas três primeiras alternativas não basta uma seleção somente sobre a tabela de monitoramento, a tabela monitorada precisa ser incluída na consulta para o caso do valor contido nela ser o valor válido no instante desejado. Todas as consultas são relativamente simples. Na Tabela 4.25 é possível verificar o desempenho das consultas.

Tabela 4.25: Desempenho de consultas de seleção temporal diretamente sobre a tabela de monitoramento

Run ID/Alternativa	SI	CI	COIA	COIP
1	2646	3869	3302	2313
2	2584	4105	3541	2013
3	2255	3456	3701	2247
4	2322	3837	3658	2217
5	2867	2990	3646	2072
6	2234	3161	3834	2407
Média	2485	3570	3614	2212

O resultado esperado para a Tabela 4.25 segue a tendência verificada nas outras consultas, com as alternativas sem inserções e com registro de operações e imagens posteriores tendo desempenho superior. Mesmo sendo semelhante à consulta de reconstrução da tabela em um ponto no tempo, se verificaram resultados diferentes dos vistos na tabela 4.21, com uma queda de performance mais amena nas alternativas com pior performance, o que mostra uma tendência de que quando aumenta o volume de dados as características de melhor ou pior desempenho tendem a ser realçadas.

Após terminados os experimentos, algumas conclusões podem ser tiradas. As consultas de seleção são executadas com maior eficiência nas alternativas com as

consultas menos complexas, a alternativa sem inserções e a alternativa com registro de operações e com imagens posteriores. Considerando apenas a performance, a alternativa sem inserções seria mais indicada. Entretanto, devido às suas restrições, as situações que permitem a sua aplicação são extremamente restritas, sendo a alternativa com registro de operações e com imagens posteriores a mais indicada para ser utilizada pelo projetista de dados na maior parte dos casos. Comparando essa alternativa com as outras duas, a restrição referente à alternativa com inserções é pouco comum e em muitos casos, contornável. Entretanto, mesmo assim essa alternativa não vale a pena se comparada com a alternativa com registro de operações e com imagens posteriores porque as características dessa alternativa facilitam a escrita de determinadas consultas e ainda ajudam em termos de performance, o que faz com o que o campo a menos na tabela de monitoramento dificilmente compense. Já comparando as duas alternativas com registro de operações, a alternativa com imagens posteriores se mostrou superior, sendo que no critério de volume de dados ela ocupará mais ou menos espaço dependendo da relação de frequência de ocorrência entre inserções e remoções.

5 CONCLUSÃO

O objetivo desse trabalho era analisar algumas alternativas para representar dados temporais em aplicações de bases de dados puramente relacionais. Para atingir esse objetivo, as alternativas foram comparadas por meio de experimentos. Através desses experimentos, foi possível chegar a algumas conclusões, amparadas pelos dados gerados pelos experimentos, o que pode ajudar projetistas e desenvolvedores de aplicações de bases de dados que envolvem dados temporais a tomar as decisões corretas em termos de design e implementação de acordo com o contexto de sua aplicação.

Nesse trabalho foram analisadas alternativas para representar o tempo de validade dos dados em duas situações diferentes encontradas em tabelas com dados temporais: atributos temporais e atemporais misturados e entidades temporais. Quando existem atributos temporais e atemporais misturados, apenas alguns atributos de uma tabela tem a variação dos seus valores ao longo do tempo registrada. Já entidades temporais se verificam quando todos os campos de uma tabela tem a variação de seus valores no tempo registrada. Além disso, foi estudada uma terceira alternativa para lidar com a variação de valores de dados no tempo, utilizando uma tabela de monitoramento de alterações através do tempo de transação para possibilitar consultas temporais e reconstrução da tabela monitorada em qualquer instante no tempo.

No estudo realizado sobre alternativas para lidar com atributos temporais verificou-se que separar os campos temporais de atemporais acaba sendo uma escolha, no geral, recomendável. Mesmo que isso exija que as consultas executem junções entre as tabelas, o menor volume de dados gerado acaba, na maioria das situações, compensando, já que o espaço em disco economizado é proporcionalmente maior à queda de desempenho apresentada em alguns tipos de consultas. Essa escolha também permite otimizar alguns tipos de consultas, como por exemplo, consultas de estado corrente, mantendo o mesmo na tabela atemporal. Além disso, a escolha de períodos para representar os dados, nesse caso, apresenta desempenho bastante superior ao uso de instantes, sendo este desaconselhado ao lidar com atributos temporais.

Já na análise sobre as alternativas para representar dados temporais referentes a entidades temporais, verificou-se que a necessidade de procedimentos externos para manter a integridade dos dados desempenha papel crucial na performance de operações de atualizações ou remoções sobre esse tipo de dados. Enquanto alternativas que utilizam apenas instantes dependem de checagens mais simples para manter a integridade dos dados, alternativas que utilizam períodos para representar a validade dos dados dependem de procedimentos mais complexos, que acabam sendo mais pesados e tendo maior impacto no desempenho das consultas. As chaves primárias em tabelas desse tipo precisam ser temporais, o que exige procedimentos que verifiquem a integridade dos dados, já que chaves primárias temporais não são suportadas por SGBDs tradicionais. Além disso, essa constatação se torna especialmente verdadeira quando a base de dados possui mais de uma tabela representando entidades temporais e elas estão envolvidas em relacionamentos que requerem o uso de chaves estrangeiras. Nessa situação, as chaves estrangeiras também precisam ser temporais, e as verificações de integridade associadas a elas são muito mais complexas quando a validade dos dados é representada utilizando períodos do que utilizando instantes.

Assim como ocorre com atributos temporais, consultas tem desempenho superior quando a validade dos dados é representada utilizando períodos. Entretanto, o desempenho em operações de atualizações e remoções é bastante inferior, afetado pela natureza dos procedimentos de verificação de integridade mencionados anteriormente. Esses resultados indicam que a escolha ideal, ao lidar com entidades temporais, vai depender muito do contexto da aplicação sendo desenvolvida. Aplicações onde inserções, e especialmente, atualizações e remoções ocorrem com frequência, não deveriam utilizar períodos para representar a validade, sob o risco de terem uma performance geral ruim. Já em um contexto onde operações que geram alterações nos dados são mais raras, o uso de períodos pode ser benéfico, já que seu desempenho em consultas é superior. Uma terceira opção que poderia ser empregada seria o uso de períodos sem o uso de procedimentos que garantam a integridade dos dados, mas isso pode ser arriscado de ser aplicado em muitas situações.

Um ponto a favor do uso de períodos para representar a validade dos dados de entidades temporais é o volume de dados menor gerado. Mesmo com um campo a mais necessário para a utilização de períodos, como essa solução não usa chaves primárias nativas da base de dados, a criação de índices fica a critério do projetista da base de dados, não sendo obrigatório a existência dos mesmos. Isso deixa o projetista mais livre para fazer um balanceamento entre espaço em disco e necessidade de melhoras de performance. Índices ocupam bastante espaço em disco, sendo possível economizar bastante nesse sentido limitando seu uso. Alternativas que utilizam instantes usam chaves primárias padrão do SGBD para garantir a integridade dos dados, e em muitos dos SGBDs tradicionais, como o PostrgreSQL utilizado nos experimentos desse trabalho, não permitem chaves primárias sem a criação automática de índices.

Por fim, no estudo realizado sobre o uso de uma tabela de registro de alterações, utilizando o tempo de transação, foram comparadas quatro alternativas para implementação da mesma. Essas alternativas vão desde uma mais simples, mas com grandes limitações (sem registrar inserções) até uma mais elaborada, que envolve não apenas registrar os valores envolvidos nas alterações mas também as operações que os geraram.

Nas análises realizadas sobre o volume de dados de cada tabela, os estudos realizados mostraram uma já esperada economia de dados ao não se recordar inserções. Os estudos mostraram também que a esparsidade da tabela de monitoramento, afetada pelo uso de imagens anteriores ou posteriores e pela frequência de diferentes operações de inserções ou remoções de dados, também influencia no tamanho da tabela monitorada em termos de volume de dados. Quando espaço em disco é um recurso escasso, o projetista deve considerar qual a frequência com que remoções e inserções de dados ocorrem, para saber se ele pode economizar espaço em disco utilizando imagens anteriores ou posteriores na tabela de monitoramento.

Em termos de desempenho, as alternativas que obtiveram melhor desempenho foram a alternativa sem registro de inserções e a alternativa com registro de operações utilizando imagens posteriores. Ambas as alternativas tiveram desempenho similar nas consultas estudadas, já que consultas mais complexas tendem a ser executadas com mais dificuldade. O bom desempenho dessas alternativas é explicado pela simplicidade de suas consultas. Na primeira, o fato de inserções não serem registradas simplifica algumas consultas, enquanto que na segunda o uso de imagens posteriores obtém o mesmo efeito. A alternativa com registro de operações, porém, não possui nenhum tipo de restrição em termos de reconstrução do estado exato da tabela monitorada em algum

instante no tempo, enquanto alternativas sem esse tipo de registro possuem limitações quando ocorrem remoções e reinserções de dados na tabela monitorada. Isso sem contar a restrição óbvia existente na alternativa sem registro de inserções, que impede que se saiba quando determinados dados foram inseridos na tabela monitorada.

5.1 Limitações e trabalhos futuros

Por envolver um campo muito amplo, alguns aspectos não puderam ser cobertos nesse trabalho, limitando a usabilidade dos dados levantados. Entretanto, essas mesmas limitações oferecem oportunidades de expansão desse trabalho através de trabalhos futuros.

Nesse trabalho não é coberta a modelagem de sistemas que envolvam tabelas bitemporais. Esse tipo de tabela aparece com frequência na literatura (SNODGRASS, 1999; ATAY, 2010; DETIENNE, 2001) e poderia fazer parte de um estudo similar ao conduzido nesse trabalho.

Além disso, também não é coberta a possibilidade de se querer garantir que um atributo temporal possua valor em todos os pontos no tempo. Por se tratar de um atributo, se entendeu que essa situação era menos comum, e portando, poderia ficar de fora do estudo conduzido. Se essa possibilidade fosse considerada, é possível que soluções que utilizam períodos sofressem decréscimo de performance, como verificado no caso das entidades temporais.

Outra possibilidade de extensão desse trabalho inclui alterar o formato dos experimentos, variando o número de registros nas tabelas e verificando a evolução dos números de cada alternativa em função disso. Considerando o número de experimentos executados e o número de alternativas cobertas, incluir esse tipo de experimentação inviabilizaria o término desse trabalho, mas essa é uma extensão interessante a ser considerada para trabalhos futuros.

Por fim, existe ainda a possibilidade de executar os experimentos utilizando diferentes SGBDs. Apesar da crença de que o fator determinante para que uma base de dados atenda ou não as necessidades das aplicações da melhor maneira possível seja a modelagem, pode ser interessante verificar o quanto os números de cada alternativa variam em diferentes SGBDs. Além disso, mesmo que no momento o suporte para operações e registro de dados temporais seja limitado e muito parecido nos SGBDs tradicionais, isso tende a mudar nos próximos anos, com a inclusão de novas estruturas no padrão SQL:2011.

5.2 Contribuições e Considerações Finais

Ao fim da realização desse trabalho, conclui-se que é muito importante combinar análises teóricas sobre diferentes alternativas de implementação para um determinado problema a análises práticas. No caso das alternativas estudadas nesse trabalho, enquanto algumas conclusões eram esperadas e algumas delas até mesmo óbvias, outras se mostraram até surpreendentes. Além disso, com dados concretos, é possível mensurar o impacto de determinadas escolhas ao invés de apenas ter uma noção do que é melhor ou pior em cada situação. Ou seja, é possível saber o quanto cada escolha será melhor ou pior em cada critério e assim poder pesar melhor os prós e contras de cada uma delas, ao compor uma solução para o problema enfrentado. Lembrando que a definição do que é melhor ou pior depende do contexto de cada aplicação, uma vez que cada

contexto tem prioridades específicas, como economia de volume ou melhor desempenho.

Essa é justamente a grande contribuição desse trabalho, prover dados concretos que suportem decisões de projeto de bancos de dados temporais sobre bases de dados puramente relacionais. Esse era também o objetivo inicial do trabalho, que acredita-se ter sido cumprido.

REFERÊNCIAS

NOH, S. Y. **An XML Representation of the Parametric Data Model for Temporal Data**. 2008. Disponível em: < http://goo.gl/F0lDw >. Acesso em out. 2012.

NOH, S. Y.; GADIA, S. K. Benchmarking temporal database models with interval-based and temporal element-based timestamping. **Journal of Systems and Software**, v.81, n.11, p. 1931-1943, nov. 2008.

BERGAMASCHI, S.; SARTORI, C. Chrono: a Conceptual Design Framework for temporal entities. In: International Conference on Conceptual Modeling, ER '98, 17., 1998. **Proceedings...** Singapore: ER '98 Press: Springer, 2000.

ESRI.; **ArcGIS Help 10.1.** 2012. Disponível em: < http://goo.gl/uq02V >. Acesso em out. 2012.

SNODGRASS, R. T.. Referential Integrity. In: SNODGRASS, R. T. **Developing Time-Oriented Database Applications in SQL**. Morgan Kaufmann Publishers, Inc., San Francisco, 1999. p. 127-129.

SNODGRASS, R. T.. Bitemporal Tables. In: SNODGRASS, R. T. **Developing Time-Oriented Database Applications in SQL**. Morgan Kaufmann Publishers, Inc., San Francisco, 1999. p. 277-340.

TANSEL, A. U. Temporal Data Modeling and Integrity Constraints in Relational Databases. In: International Symposium on Computer and Information Sciences, ISCIS'04, 19., 2004. **Lecture Notes in Computer Science.** Antalya, Turkey: ISCIS'04 Press: Springer, 2004.

SNODGRASS, R.T. The temporal query language TQuel. **ACM Transactions on Database Systems**, v.12, n.2, p. 247–298, jun. 1987

MKAOUAR, M., BOUAZIZ, R., AND MOALLA, M. Querying and Manipulating Temporal Databases. **International Journal of Database Management Systems (IJDMS)**, v.3, n.1, fev. 2011

MORO, M. M., EDELWEISS, N., SANTOS, C. S. Modelo Temporal de Versões. R**ITA**, v.9, n.2, jan. 2002

WANG, F.; ZANIOLO, C. ; ZHOU, X. ArchIS: an XML-based approach to transaction-time temporal database system. **The VLDB Journal — The International Journal on Very Large Data Bases**, v.17 n.6, p.1445-1463, nov 2008

SNODGRASS, R. T. **Temporal databases status and research directions**, ACM SIGMOD Record, v.19 n.4, p.83-89, Dec. 1990 [doi>10.1145/122058.122068]

MELTON, J. 1996. **SQL/Temporal**. Tech. Rep. ISO/IEC JTC 1/SC 21/WG 3/DBL-MCI-012.

SNODGRASS. R; **TSQL2 and SQL3 Interactions** 2012. Disponível em: < http://www.cs.arizona.edu/~rts/SQL3.html >. Acesso em dez. 2012

MELTON, J.; SIMON, A.R. 1993. **Understanding the New SQL: A Complete Guide.** San Francisco, CA: Morgan Kaufmann.

AHMED, E. B., NABLI, A., GARGOURI, F. , New Algorithm for Cyclic Mining in Temporal Databases. **Journal of Applied Sciences**, 2012

ALI, N. H., SULAIMAN, S. Managing News Archive Using Temporal Data Modeling. **Journal of Applied Sciences**, 2012

KULKARNI, K., MICHELS, J. E. Temporal features in SQL:2011. **ACM SIGMOD Record**, Volume 41 Issue 3, Pages 34-43, Set. 2012

ATAY, C. E. A Comparison of Attribute and Tuple Time Stamped Bitemporal Relational Data Models, International Conference on Applied Computer Science. **Proceedings...** Out. 2010

O'CONNOR, M. J., TU, S. W. , MUSEN M.A. The Chronus II temporal database mediator. AMIA Symposium. **Proceedings...** 2002

GUNADHI, H.; SEGEV, A. A framework for query optimization in temporal databases. International conference on statistical and scientific database management. **Procedings...** 1989

PARENT, C., SPACCAPIETRA, S., ZIMÁNYI, E. Spatio-temporal conceptual models: data structures+ space+ time. 7th ACM international symposium on Advances in geographic information systems. **Procedings...** 1999

HOBBS, J. R., PAN, F. An ontology of time for the semantic web, **ACM Transactions on Asian Language Information Processing (TALIP)**, v.3 n.1, p.66-85, mar. 2004 [doi>10.1145/1017068.1017073]

TANSEL, A. U. 2008. **Temporal Databases**. Wiley Encyclopedia of Computer Science and Engineering. 1–7.

ALLEN, G. N., MARCH, S. T. **Temporal Database Management and the Representation of Temporal Dynamics**. Univeristy of Minnesota Working Papers 2000. Abr. 2000

MYSQL; **MYSQL 5.1 Reference Manual.** 2012. Disponível em: < http://dev.mysql.com/doc/refman/5.1/en/storage-requirements.html >. Acesso em dez. 2012

SCHIEFER, K. B. , VALENTIN G. - DB2 universal database performance tuning. **Data Engineering Bulletin**, 1999

GRAEFE, G., SHAPIRO, L.D. Data Compression and Database Performance. **Applied Computing**, 1991

POSTGRESQL. **PostgreSQL documentation**. 2012. Disponível em:< http://www.postgresql.org/docs/8.4/static/indexes-unique.html>. Acesso em dez. 2012

DETIENNE V. ,HAINAUT J. CASE tool support for temporal database design, 20th International Conference on Conceptual Modeling. **Proceedings...** ER'01, LNCS 2224, Springer-Verlag, Yokohama, Japan (2001), pp. 208–224

MANICA, E. , CERVI, C. R. , DORNELES, C. F. , GALANTE, R. **Ferramenta para Suporte a Consultas Temporais em SGBDs convencionais.** In: V Escola Regional de Banco de Dados (ERBD), April 2009, Ijuí, RS, Brazil.

SNODGRASS, R. T. **The TSQL2 Temporal Query Language**. Kluwer Academic Publisher, 1995

GREGERSEN, H., JENSEN, C. S. : **Temporal Entity-Relationship Models - A Survey**. IEEE Trans. Knowl. DataEng. 11(3): 464-497 (1999)

TORP, K., JENSEN, C. S., BÖHLEN, M. H. **Layered Temporal DBMS: Concepts and Techniques.** DASFAA 1997: 371-380

TORP, K., JENSEN, C. S., RICHARD T. SNODGRASS **Stratum Approaches to Temporal DBMS Implementation.** IDEAS 1998: 4-13

TORP, K. **Temporal Strata.** Encyclopedia of Database Systems 2009: 3018-3023

JOHNSTON T., WEIS R. **Managing Time in Relational Databases.** Morgan Kaufmann, 2010