Um Sistema de Pre-processamento de atualizações em bancos de dados relacionais

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Liliane Leopoldine d'Oliveira

e aprovada pela Comissão Julgadora.

Campinas, 13 de junho de 1989

Prof.Dr. _____
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de Mestre em CIÊNCIA DA COMPUTAÇÃO

UM SISTEMA DE PRÉ-PROCESSAMENTO DE ATUALIZAÇÕES EM BANCOS DE DADOS RELACIONAIS

Liliane Leopoldina D'oliveira

ORIENTADOR: Prof. Dra. Claudia Bauzer Medeiros

março 1989

CONTEÚDO

íNDICE DE FIGURAS

RESUMO

A tese apresenta um sistema para condensação e diferimento de atualizações em bancos de dados relacionais.

O sistema permite controle de atualizações diferenciais e pode ser utilizado tanto para pré-processamento de atualizações em relações quanto para "refresh" de instantâneos.

O sistema implementado pode também ser adaptado para permitir manutenção de cópias e fragmentos de relações em nós de uma rede.

A implementação foi feita em PASCAL 3.4, em interação com SGBD relacional RDB, em um sistema VAX 11/785.

AGRADECIMENTOS

Este trabalho não teria sido possível sem o apoio e o incentivo de minha orientadora. Assim eu não poderia deixar de agradecer à Claudia pela orientação, paciência e amizade durante esses anos de estudo e trabalho. Gostaria ainda de agradecer ao Prof. Dr. Geovane Cayres Magalhães, pelo fornecimento de material bibliográfico e esclarecimentos, ao Prof. Dr. Hans Kurt Edmund e ao Prof. Dr. Célio Cardoso Guimarães pelas sugestões, aos meus colegas de trabalho e do curso de pós-graduação, as bolsas recebidas da CAPES E IBM - Brasil e aos meus pais.

Campinas, 28 de março de 1989.

1. INTRODUÇÃO.

Esta tese se propõe a analisar os problemas de pré-processamento e de diferimento de atualizações em bancos de dados relacionais, propondo uma maneira de solucioná-los.

O pré-processamento de operações de consulta ou atualização em bancos de dados é uma técnica que tem como objetivo reduzir o número de acessos a arquivos, requisitados por transações.

O problema de tráfego de dados se faz sentir ainda mais quando informações são armazenadas ao longo dos nós de uma rede de comunicação de dados, caso em que o pré-processamento racionaliza o tráfego de dados na rede. Quando este pré-processamento é aliado ao diferimento das operações, permite que estas sejam realizadas mesmo quando não é possível executá-las imediatamente (por exemplo, se há queda do sistema no nó a ser acessado).

A grosso modo, pré-processamento de atualizações consiste em analisar pedidos de atualizações à medida em que são solicitados, antes de proceder à sua execução. Um dos objetivos é identificar quais atualizações podem ser eliminadas; outro objetivo é determinar quais podem ser condensadas ou reformuladas, para maior eficiência de processamento. Essa técnica pode ser aplicada tanto durante períodos pré definidos de trabalho (que podem inclusive ser especificados pelo usuário), como ser ativada através de gatilhos acionados por eventos específicos (por exemplo quando há queda de nó em sistemas de bancos de dados distribuídos). No último caso, o pré-processamento é associado às operações padrão de "logging", de forma a permitir atualização do nó quando reconectado à rede. Uma outra aplicação é a de atualizações sob demanda [ROU 1986]:

Atualizações são efetuadas quando um objeto é consultado e as operações de atualização podem ser então otimizadas.

O pré-processamento de operações é um problema encontrado sob vários prismas na literatura, tanto para operações de consulta como para atualização. Este capítulo contém uma análise das tendências na área. Como a ênfase da tese é sobre atualização, a seção que trata de consultas recebeu pouca atenção, havendo sido incluída apenas para indicar exemplos do tipo de tratamento dado ao problema.

Na área de pré-processamento e diferimento de atualizações em bancos de dados, há menos trabalho na literatura, sendo essa uma das motivações da tese. No processamento de operações em bancos de dados distribuídos, a localização de dados e os algoritmos de acesso têm um papel importante, merecendo uma seção à parte neste capítulo. Finalmente, a última parte deste capítulo descreve a organização da tese.

1.1 PRÉ-PROCESSAMENTO DE CONSULTAS.

O objetivo do pré-processamento de consultas é o da otimização do processamento, de forma a obter um resultado de maneira mais econômica em termos de tempo ou espaço.

O pré-processamento de consultas é comumente analisado no contexto de bancos de dados distribuídos.

O livro editado por KIM et al [KIM 1985], sobre processamento de consultas em banco de dados, contém vários artigos que discorrem sobre o tópico, tanto do ponto de vista de

bancos de dados distribuidos [LOH 1985, YU 1985] quanto do ponto de vista de pré-processamento de conjunto de consultas [JAR 1985,KIM 1985a]. Neste último caso, são analisadas consultas múltiplas (ou seja, grupos de consultas) de forma a permitir sua otimização global. Uma das técnicas, por exemplo, é a de identificar sub-expressões comuns, cujo resultado só é calculado uma única vez, sendo a partir daí utilizado por várias consultas, diminuindo assim seu tempo de processamento.

CERI E PELAGATTI [CER 1985] em seu livro fazem um excelente estudo das técnicas mais comuns de pré-processamento de consultas. Em seu texto apresentam o uso de junção e semi-junção como técnicas comuns de processamento de consultas em bancos de dados relacionais. Para otimizar consultas, apresentam métodos que estão baseados em três fases distintas: um pré-processamento local é executado; depois, semi-junções são usadas para reduzir os tamanhos das relações que são enviadas para uma posição comum; e finalmente a consulta é processada sobre as relações reduzidas.

Uma outra maneira de otimizar consultas analisada pelo livro consiste em alternar operações locais e algumas transmissões de dados. Assim, uma consulta envolvendo uma junção de três relações que estão armazenadas em nós diferentes poderá ser solucionada enviando a primeira para o local da segunda. Em seguida, faz-se a junção das duas, sendo então enviado o resultado para a terceira relação, ou seja, a consulta é pré-processada em partes visando minimizar a transmissão de dados para o processamento final [CER 1985].

A escolha entre junção e semi-junção para processamento de consultas depende também da hipótese sobre o custo relativo de transmissão e processamento local. No geral, a vantagem de usar semi-junção é maior se o custo de transmissão é considerado importante e o custo de processamento local pode ser praticamente

desconsiderado. Em contrapartida, o uso de junções é frequentemente mais conveniente que o uso de semi-junções se for considerado também o custo local de processamento na avaliação da estratégia alternativa de processamento de consultas.

Dentro de pré-processamento de consultas pode-se considerar também o processo de otimização. Em muitos casos, a otimização consiste na análise da consulta visando remover operações redundantes ou combinação (condensação) de operações. Outras otimizações buscam aproveitar características do processador de consultas ou escolher o método de avaliação. O processo de semi-junções se encaixa na categoria de otimização através de ordenamento adequado da avaliação de sub-consultas. O livro de MAIER [MAI 1983] discute o tratamento formal de decomposição, otimização e pré-processamento de consultas em bancos de dados relacionais.

Uma consulta pode também ser otimizada pela re-ordenação de suas operações ou pela utilização de algoritmos adequados na avaliação das operações.

LEE e YU [LEE 1987], por exemplo, propõem um método de compilação de consultas no qual muitas das decisões sobre seleção de caminho de execução são determinadas em tempo de execução. Em outras palavras, em vez da estratégia de execução da consulta ser determinada durante a compilação, a estratégia é influenciada por informações como número de tuplas de uma relação no momento da consulta.

Finalmente, o pré-processamento e otimização de consultas pode incluir operações de atualização, com aplicação de arquivos diferenciais [CEL 1984, ROUS 1986]. Este tipo de enfoque será visto mais adiante e tem por objetivo reduzir o "overhead" no processamento de atualização.

Os textos indicados nesta seção fornecem uma boa visão geral da área de otimização de consultas, processo em que o pré-processamento é comum .

1.2 PRÉ-PROCESSAMENTO DE ATUALIZAÇÕES.

Esta seção descreve situações na literatura em que é dada atenção ao pré-processamento e diferimento de atualizações, quer para manutenção de instantâneos [LIN 1986] e processamento diferencial de atualizações [KAH 1987, BLA 1986], quer para ativação de gatilhos [BUN 1982] ou na manutenção de cópias de relações em nós de bancos de dados distribuídos [DAN 1983, COO 1984]. O capítulo também descreve técnicas de se manter logs para processamento de atualizações.

1.2.1 ATUALIZAÇÕES DIFERENCIAIS: MANUTENÇÃO DE INSTANTÂNEOS E VISÕES MATERIALIZADAS.

Um instantâneo de um banco de dados é um arquivo para leitura sómente, cujo conteúdo é extraido dos arquivos do banco de dados e é atualizado periódicamente para refletir o estado atual do banco de dados. Alguns programas aplicativos podem fazer consultas sobre o estado corrente do banco de dados e modificá-lo. No entanto, muitas aplicações não têm necessidade do estado

corrente podendo utilizar porções "congeladas" do banco para, por exemplo, análises estatísticas ou planejamento. Afim de manter aplicações que requerem uma versão estática de parte do banco de dados, os dados relevantes podem ser copiados em arquivos que não fazem parte do banco de dados próprìamente dito. Isso permite que o banco de dados continue a refletir uma visão consistente do mundo real, enquanto cópias estáticas (os instantâneos) são mantidas para aquelas aplicações que as necessitem.

Em bancos de dados relacionais, um instantâneo é uma tabela (relação) sómente para leitura, cujos valores são definidos através de consultas sobre uma ou mais relações. Um instantâneo pode ser atualizado para refletir o estado corrente das tabelas do banco de dados. Esta operação é denominada "refresh". Além disso, podem ser utilizados ao invés das tabelas a partir das quais foram derivados, por aplicações que não requerem acesso ao estado atual, e nem precisam atualizar o estado corrente. Em bancos de dados distribuídos, instantâneos locais ou remotos podem ser atualizados periódicamente a partir de relações residentes em outros nós. Uma vez que o instantâneo tenha sido definido e inicializado, seus dados podem ser acessados usando consultas simples e funcionam como arquivos do bancos de dados. Por exemplo, índices podem ser definidos sobre o instantâneo para acelerar o acesso ao seu conteúdo e instantâneos podem servir como base para geração de outros instantâneos.

Um outro tipo de objeto derivado que precisa ser mantido atualizado de forma a refletir modificações em relações base é a chamada "visão materializada" [BLA 1986, BLA 1987]. Enquanto visões são resultado de consultas, gerada a cada solicitação do usuário, visões materializadas são arquivos não voláteis que resultam do armazenamento de alguma visão. Ao contrário de instantâneos, que podem ser atualizados periódicamente ou então

regerados, visões materializadas devem sempre refletir o estado atual das relações base.

Um dos primeiros trabalhos sobre manutenção de instantâneos [SEV 1976] sugere a utilização de arquivos diferenciais. A cada relação no banco de dados é associado um arquivo que contém todas as modificações recentes na relação desde o último "refresh". Este arquivo contém todas as tuplas inseridas e eliminadas da relação. O "refresh" é então processado pela execução das atualizações correspondentes no instantâneo.

A quantidade de informações transferidas para o instantâneo durante a operação de atualização pode ser reduzida se a tabela base não foi modificada substancialmente desde a última atualização do instantâneo. Todas as mudanças que ocorreram desde a última atualização devem ser detectadas e aplicadas quando afetarem o instantâneo.

Uma vez que instantâneos podem ser vistos como um sistema de tabelas, a atualização destes pode ser facilmente conseguida, reconstruindo o instantâneo a partir de suas tabelas base. Essa estratégia é chamada de "atualização integral". No entanto, se poucas ou nenhuma modificação é feita nas tabelas base envolvidas na definição do instantâneo desde seu último "refresh", muitas das operações de atualização serão redundantes.

Em oposição à atualização integral, a estratégia da "atualização diferencial" está baseada na detecção de modificações feitas em cada tabela base envolvida na definição do instantâneo desde o último "refresh" . Assim, combinando essas modificações, operações de atualização são computadas e enviadas para o instantâneo.

Em [LIN 1986], quando o conteúdo de um instantâneo é formado por operações de "restrição" (isto é, "seleção") e "projeção"

sobre uma única relação base (original), técnicas de atualização diferencial podem reduzir o custo da operação de atualização do instantâneo. O algoritmo apresentado registra as mudanças ocorridas desde a última atualização do instantâneo para aplicá-las posteriormente. Assim, ao invés de se proceder à costumeira substituição integral do instantâneo, apenas algumas de suas partes são modificadas. A fim de prover uma manutenção eficiente para instantâneos remotos, o algoritmo procura transmitir o mínimo de dados possível durante a operação de "refresh".

O algoritmo permite também a manutenção de múltiplos instantâneos sobre uma única relação base. Cada instantâneo é atualizado independentemente, devendo especificar suas próprias restrições e projeções sobre a relação base. Isso permite que cada instantâneo extraia somente os dados necessários da relação base.

O algoritmo parte da hipótese de que as entradas da relação base estão ordenadas e embutidas num espaço de endereçamento denso. Cada elemento deste espaço ou contém uma entrada na tabela base ou é marcado como vazio. Além disso a relação base para cálculo do instantâneo possui um campo para cada atributo, onde é armazenado o tempo ("timestamp") que indica quando foi a última vez que aquele elemento foi atualizado. O instantâneo possui entradas que incluem um campo contendo o endereço da entrada correspondente na tabela base e um campo indicando quando o instantâneo foi atualizado pela última vez ("snaptime"). Embora o autor em sua introdução mencione que considera as operações projeção e seleção, trata apenas de instantâneos gerados por seleção (por simplificação foram ignoradas as projeções).

O algoritmo utiliza como ponto de partida a atualização mais recente do instantâneo (snaptime). Cada elemento da tabela base é examinado: se seu "timestamp" é maior que o "snaptime" do

elemento correspondente no instantâneo, o elemento é transmitido para o instantâneo; se o elemento é vazio ou se seu valor não satisfaz à condição de seleção, somente o endereço do elemento e o status vazio são transmitidos para o instantâneo. Embora este último tipo de operação possa parecer desnecessário, isso acontece porque entradas que não satisfazem a restrição atual do instantâneo poderiam satisfazê-la antes da modificação. Caso a restrição seja satisfeita, o endereço, o status (OK) e o novo valor são enviados para o instantâneo. Depois de percorrer a tabela base, e transmitir os elementos que foram modificados, a marca do tempo (da tabela base) corrente é enviada para o instantâneo e passa a ser o seu novo "snaptime" .

Esse algoritmo detecta todas as mudanças na tabela base e as informa ao instantâneo. Conforme mencionado, quando a função de restrição que gera o instantâneo corresponde a uma redução da tabela base, o algoritmo envia entradas supérfluas para o instantâneo. Se entradas da tabela base que não satisfazem a restrição do instantâneo são eliminadas, inseridas ou modificadas, os endereços e status são transmitidos para o instantâneo.

KAHLER [KAH 1987] apresentam dois métodos para guardar as modificações feitas em uma relação base para posteriormente proceder à atualização de um instantâneo : um log sequencial e um log condensado. Os métodos foram submetidos a testes com várias frequências e composições de atualizações. Os resultados mostram que o log sequencial possui um bom desempenho quando o instantâneo é único e o conjunto de modificações é pequeno em relação ao tamanho da tabela base, ou se o instantâneo é gerado por seleção. Para o caso de instantâneos duplicados e um número

maior de modificações, o método do log condensado é preferível.

O mecanismo de "atualização diferencial" de [KAH 1987] foi projetado para satisfazer as seguintes situações:

1) Manutenção de instantâneos que são cópias de relações ou gerados através de operações de seleção sobre uma relação;

2) Manutenção de um ou mais instantâneos (replicados);

3) Suporte a instantâneos independentes sobre uma tabela.

Os autores concentram a discussão sobre instantâneos baseados em uma única tabela base, isto é, sem consultar junção na definição do instantâneo.

Para calcular o desempenho de acesso a instantâneos, existem vários fatores a considerar. Primeiro, existe o possível "overhead" das atualizações sobre a tabela base (ou seja inserções, modificações e eliminações), que é o custo de processamento no local da tabela base. Existe também o custo de comunicação de mensagens de atualização para o nó onde reside o instantâneo e o do processamento do "refresh". Os dois últimos dependem do número de mensagens enviadas do nó onde reside a tabela base.

Trabalhos anteriores ao de [KAH 1987] têm se concentrado na minimização do envio de mensagens quando da atualização do instantâneo. No entanto, como observam os autores, não pode ser ignorado o custo do processamento local em consultas distribuidas. Enquanto a estratégia funciona na maioria dos casos, existem situações em que isso pode levar a custos inaceitáveis. Isso é especialmente importante se considerarmos múltiplos instantâneos definidos sobre uma tabela base.

Um elemento chave de muitos dos algoritmos de pré-processamento e diferimento de operações encontrados na literatura é o log, usado para armazenar modificações da tabela

base. Um log mantém, além de outras informações, registros com todas as modificações do banco de dados desde o último "backup", visando manutenção de integridade.

Conforme já mencionado, o trabalho de Kahler [KAH 1987] menciona dois tipos de log possíveis (sequencial e condensado), que serão discutidos aqui, já que dão uma boa idéia de pesquisas na área. O sistema implementado e discutido nesta tese adota algumas idéias do log condensado.

O log sequencial de [KAH 1987] é tal que cada tabela base possui o seu próprio log de atualizações organizado sob forma sequencial. Este log é utilizado para cada "refresh" de instantâneo. O "refresh" é basicamente feito enviando todas as modificações (ou seja o log) para o nó onde se encontra o instantâneo e refazendo-as sobre o instantâneo. Depois do "refresh", o log é eliminado.

O log sequencial impõe algum "overhead" sobre o processamento normal da tabela base. Quando uma tupla é inserida, modificada ou eliminada, é criada uma entrada no log refletindo a atualização feita na tabela base. A atualização em si pode ser registrada como uma entrada contendo imagens da tupla antes e depois da atualização, ou somente a imagem posterior. Se a operação for de inserção ou de modificação, a tupla é adicionada ao log. No caso de eliminação a chave primária da tupla é colocada no log. Cada entrada no log possui um rótulo que identifica o tipo de atualização.

Como os registros do log sequencial contêm todas as modificações feitas na tabela base desde a última atualização, o custo da atualização do instantâneo pode ser diminuido em termos de custo de processamento. O log sequencial é percorrido e para cada entrada no log, uma mensagem de atualização é enviada para o

instantâneo se e somente se a entrada no log for de interesse (isto é, corresponde a tupla que contribui para a formação do instantâneo).

Sómente as entradas do log do tipo inserção podem ter sua validade verificada previamente sem consulta ao instantâneo usando a definição do instantâneo para verificar se a nova tupla afeta o instantâneo. Essa verificação não pode ser feita para entradas do tipo modificação e eliminação enquanto o valor anterior da tupla da relação base for desconhecido. Como consequência, todas as modificações e eliminações devem ser enviadas para o instantâneo. O processo de "refresh" deve, portanto, estar preparado para manipular (e recusar) atualizações de tuplas não presentes no instantâneo. Modificações e eliminações "estranhas" no entanto não provocam uma atualização incorreta do instantâneo; elas apenas causam um "overhead" desnecessário.

A situação pode ser remediada salvando-se o valor anterior de cada tupla antes da modificação. Como se verá mais tarde, esta é a opção adotada na tese, de forma modificada. No caso de operações de eliminação, a tupla é integralmente escrita no log (ao invés da chave primária sómente). No caso de modificação, a imagem anterior é colocada no log, imediatamente seguida pelo novo valor da tupla. Desse modo o processo de "refresh" pode agora descartar todas as entradas no log que não satisfazem as condições de geração do instantâneo da seguinte maneira:

-inserções que não satisfazem a condição de seleção não são enviadas;

-modificações que não satisfazem a restrição nem antes e nem depois da atualização não são enviadas. Todas as outras modificações serão enviadas através de mensagens de inserção,

modificação ou eliminação dependendo dos valores anterior e posterior.

-eliminações que não satisfazem as restrições são ignoradas, e as demais são enviadas através de mensagens de eliminação com o valor da chave primária.

Um log sequencial que mantém valores anteriores e posteriores a modificação requer mais memória. No entanto o número de mensagens pode ser reduzido significativamente para o caso de instantâneos gerados a partir de seleções.

Anteriormente a essa solução, atualizações ainda que não satisfazendo as condições de geração do instantâneo, eram enviadas e cada modificação era considerada separadamente. No log sequencial, várias atualizações para uma única tupla resultam em várias entradas no log. Uma tupla que é atualizada várias vezes e depois eliminada dá origem a várias mensagens de modificação e uma mensagem de eliminação. Idealmente, sómente uma mensagem de eliminação é necessária. No entanto isso não pode ser determinado sem percorrer as entradas do log.

O histórico de mudanças para cada tupla (representado por uma sequência de modificações) pode ser condensado em uma única modificação (uma atualização seguida por outra atualização seguida de uma eliminação resulta numa eliminação, etc.). Uma vez que sómente o resultado final é necessário para o "refresh" do instantâneo, sómente este é enviado. Este tratamento corresponde à solução utilizando um log condensado.

Um log condensado tem acesso por organização indexada a partir de um identificador único (por exemplo chave primária). Cada entrada do índice aponta para a modificação (condensada) da tupla desde a última atualização. O tamanho do log de modificações é reduzido e gravações e leituras intermediárias são totalmente eliminadas.

Por exemplo se nenhuma entrada de modificação é encontrada para uma tupla, isto é, não existe nenhum pedido anterior de atualização, então a modificação em si é salva, ou seja, é a primeira modificação feita para a tupla desde o último "refresh" do instântaneo.

Para superar o problema de incorporar todas as eliminações e modificações no envio de mensagens de atualização para o instantâneo, o valor antigo da tupla é salvo antes da sua primeira modificação depois do "refresh" do instantâneo. O log condensado é similar ao log sequencial contendo imagens anteriores e posteriores. Existe, no entanto, algumas diferenças: uma modificação de uma tupla é representada por uma única entrada no log. No log sequencial, uma entrada de modificação possui os valores anterior e posterior à atualização, a cada modificação. No log condensado, a imagem antes da atualização corresponde ao valor anterior à primeira modificação.

O "overhead" resultante da condensação de modificações de tuplas permite que o processo de atualização local determine se uma tupla modificada foi incluida no instantâneo, uma vez que somente imagens anteriores de tuplas que satisfazem os critérios da definição podem ser armazenadas no instantâneo. Atualizações e eliminações que não afetam o instantâneo podem ser descartadas aplicando as regras descritas para a estratégia sequencial revisada. Ao contrário da estratégia sequencial, mensagens de atualizações "estranhas" podem ser evitadas. Segundo [KAH 1987], para muitos instantâneos gerados a partir de seleções, isso pode resultar numa boa economia de tráfego de dados.

É possível que mais de uma cópia de um instantâneo possa existir sobre uma tabela base (instantâneos replicados). Além disso, diferentes instantâneos podem ser definidos sobre uma

tabela base (instantâneos independentes). A principal diferença entre as duas formas é que, na primeira, o usuário desejará que todas as cópias de um instantâneo replicado sejam atualizadas concorrentemente, ao contrário da segunda, onde os instantâneos possuirão sua própria frequência e atualizações independentes.

[KAH 1987] analisa a questão de quão tolerante deve ser o processo de atualização das réplicas residentes em nós desconectados devido a falhas, que ocorrem antes ou durante o processo de atualização. Se o nó torna-se indisponível para atualização, existem duas possibilidades:

- abandonar a atualização até que todos os nós envolvidos estejam disponíveis. Porém essa solução diminuiria a disponibilidade das informações "atualizadas" nos instantâneos.

- continuar atualizando os nós restantes, se uma perda de consistência de uma cópia pode ser tolerada. Nesse caso é preciso definir qual o estado da tabela base que deve ser refletido quando os nós em questão forem recuperados. De novo, existem duas alternativas: todas as cópias deverão refletir o estado no tempo da chamada do processo de "refresh", ou cada réplica pode refletir o estado mais atual da tabela base.

O exemplo a seguir é retirado do mesmo artigo, por ser uma boa ilustração de como proceder ao "refresh" de instantâneos replicados quando algum nó onde reside uma das cópias do instantâneo em questão não está disponível.

Considere uma mudança de produto que produz uma atualização no catálogo de produtos de uma determinada empresa. Sejam todas as cópias de instantâneos de catálogos: S1, S2,...,Sn que estão no mesmo estado. Um novo produto, digamos P1, é inserido no catálogo. No momento do "refresh", o nó que possui S1 não está disponivel. O fato é ignorado e as cópias restantes são

atualizadas. Em seguida, um novo produto P2 é inserido. Quando o nó de S1 estiver disponível novamente, deve ser imediatamente atualizado, segundo duas opções: a) P1 e P2 serão colocados no instantâneo; ou

b) sómente P1 será inserido esperando a próxima atualização para P2 ser adicionado em todas as cópias. Se esta última for escolhida, informações necessárias para regerar o estado no instante da chamada do processo de "refresh", deverão ser mantidas por algum tempo, marcando apropriadamente novas mudanças para serem distinguidas das anteriores.

Considerações similares devem ser feitas para instantâneos independentes. Como o problema de réplicas é frequente no instante da chamada do procedimento de "refresh" e no instante do diferimento de atualização, instantâneos independentes possuem diferentes tempos de chamada de atualização/diferimento de atualização.

Para manter instantâneos independentes, o log sequencial não pode ser logo descartado depois do "refresh" de um instantâneo. As entradas do log serão mantidas até que todos os instantâneos definidos sobre a tabela base tenham sido atualizadas corretamente. A fim de evitar a pesquisa integral da tabela base para um particular "refresh" de um instântaneo, um mecanismo é usado para identificar o instante do último "refresh" de cada instantâneo. Cada instantâneo será associado com a "marca de atualização" no log, identificando a entrada atualizada mais recentemente para um instantâneo. Na ocorrência de um novo "refresh", sómente entradas no log com a marca precisam ser consideradas. Entradas já examinadas para todos os instantâneos definidos sobre a tabela base (as "marcas de atualização" de menor valor) podem ser descartadas.

Atráves da associação da "marca de atualização" com cada

instântaneo, refletindo o intante de seu último "refresh", o algoritmo pode suportar instantâneos independentes. Esta extensão permite também manter instantâneos replicados no caso onde ele é suficiente para permitir que cada cópia reflita o estado da tabela base no instante da atualização e não no instante da chamada do procedimento de "refresh".

Neste caso, cada cópia é simplesmente tratada como se fosse um instantâneo independente, sendo que a "marca" deve ser mantida e associada a cada réplica.

Os dois métodos (log sequencial e o log condensado) de [KAH 1987] são utilizados para a atualização diferencial de instantâneos baseados numa tabela separada para um log de atualizações feitas para uma tabela base.

Se o instantâneo está replicado em vários nós, ou se muitos instantâneos independentes são definidos sobre a tabela base, então o log condensado é preferível.

Um sistema pode suportar ambos os métodos. O administrador do banco de dados pode então escolher um deles, como por exemplo utilizar o método sequêncial quando a maioria das atualizações se referem a poucas tuplas, ou mesmo decidir dinâmicamente a mudança do método como por exemplo se a tabela for vazia optar pelo uso de "refresh integral".

Ambos os métodos podem ser usados para suportar outras facilidades como visões instanciaveis e mecanismos de atualiza ções diferidas.

Todas estas sugestões para manutenção de vários instantâneos acarretam obviamente maior espaço ocupado pelo log, que passa a ter que contar com marcas e modificações temporais.

O problema de aplicação e propagação de atualizações no caso de relações replicadas é chamado por EAGER e SEVCIK [EAG 1983] de

problema da "atualização perdida" (missing update). O contexto não é o de instantâneos, mas as soluções discutidas (como e quando um nó deve fazer a propagação de atualizações) pode ser aplicado a esta discussão. Uma atualização é dita "perdida" quando uma transação não pode atualizar uma cópia de uma relação por queda do nó. Neste caso, outros nós podem manter a informação (diferencial) a ser enviada quando da restauração do nó.

Outro trabalho sobre atualização de instantâneos foi desenvolvido a partir de "visões materializadas" ou "visões instanciaveis" [BLA 1986]. Uma visão de um banco de dados é instanciada sob a forma de uma tabela, no lugar de uma definição que seria avaliada toda vez que fosse referenciada numa consulta. A manutenção de visões instanciáveis é sugerida para ser executada de forma diferencial. É apresentado um método através do qual se pode deduzir modificações necessárias na visão a partir de atualizações de tabelas referenciadas pela visão.

[BLA 1987] estende o trabalho anterior de pré-processamento e diferimento de atualizações em bancos de dados relacionais de [BLA 1986]. Seu trabalho é baseado num ambiente que pressupõe um conjunto de relações base que é utilizado para criar arquivos (visões) em vários nós de um sistema distribuido. Apresentam algoritmos que determinam que fragmentos de relações base devem ser enviados a quais nós para permitir a manutenção de visões materializadas, dos arquivos, atualizadas. O trabalho é associado a uma teoria que determina quando relações são irrelevantes no cálculo de fragmentos para transmissão, mesmo quando estas relações servem de base a visões. A diferença de tratamento para instantâneos é que os autores supõem que visões materializadas devem ser modificadas imediatamente, ao contrário do processo intermitente de "refresh" de instântaneos. Os resultados podem , no entanto, ser utilizados neste último tipo de aplicação.

O trabalho de [BLA 1987] voltará a ser discutido posteriormente nesta tese, para mostrar como as estruturas de dados aqui propostas podem servir para diferimento de atualização de instântaneos e processamento de visões materializadas.

1.2.2 DIFERIMENTO E GATILHOS.

Quando atualizações são diferidas, o momento de sua aplicação é determinado por gatilhos inseridos por programadores ou embutidos no sistema.

A utilização de gatilhos para processamento de operações diferidas é exemplificada em BUNEMAN E CLEMONS [BUN 1982]. Os autores apresentam um programa denominado de "alerter" que monitora um banco de dados e informa ao usuário ou a um programa quando uma condição específica ocorre. Associar um "alerter" a uma condição que envolve operações sobre várias relações é equivalente a associá-lo a uma relação virtual que representa a condição (semelhante a uma visão). Alerters são usados, entre outras coisas, para monitorar atualizações. O problema de determinar quando ou não um "alerter" será ativado consiste em certificar-se de que uma atualização no banco de dados não afeta a relação virtual, ou afeta de uma maneira irrelevante o "alerter" (a atualização é pré-processada).

A estratégia consiste em transformar a condição que ativa um alerter em expressões de álgebra relacional. Para cada uma das operações desta expressão é verificado qual o efeito sobre uma dada relação virtual gerada pela expressão. Estes efeitos são gerados pelo algoritmo progressivamente à medida em que as

operações são efetuadas.

Um outro sistema de diferimento de atualização associado a gatilhos .é proposto por [CAS 1988], no contexto de manutenção de integridade de relações sujeitas a restrições de integridade referencial. Os pedidos de atualização são pré-processados por um monitor, que determina que atualizações devem ser propagadas. A cada propagação necessária, o sistema cria entradas em uma tabela de propagações, sem, no entanto, processá-las. Terminada a sessão, a tabela é analisada e são acionados gatilhos necessários para restaurar a consistência global. As entradas da tabela mantêm informação suficiente para acionamento dos gatilhos de forma a evitar propagação recursiva de atualizações e facilitar o processamento do monitor.

Em alguns casos, o gatilho que aciona uma atualização diferida é a solicitação de consulta do dado. CELIS [CEL 1984] descreve um projeto conceitual de um sistema de diferimento de atualizações para um banco de dados relacional. A técnica consiste em adiar a atualização de qualquer tupla até que esta seja acessada para leitura.

O algoritmo mantém relações generalizadas (uma forma mais compacta de representar a relação original) na memória principal. Atualizações são registradas nessas relações. No momento em que essas tuplas forem acessadas pelo usuário para leitura, essas relações serão pré-processadas e só então o banco de dados será modificado.

Outro exemplo de diferimento de atualização à espera de uma consulta é encontrado em [ROU 1986]. Os autores utilizam o que chamam de estratégia "preguiçosa" de atualização. A manutenção de visões e objetos é diferida até que sejam requisitados. Este procedimento visa diminuir o "overhead" de atualizações.

Utilizam, nesse sistema, um algoritmo de atualização incremental que propaga atualizações de visões para visões intermediárias até atingir as relações base usadas na materialização de uma visão. O algoritmo se baseia em um conjunto de arquivos diferenciais e as atualizações são efetuadas sobre índices especiais usados para acessar (partes de) uma visão.

O custo de processamento de uma atualização é considerado nulo, já que as operações realizadas são as mesmas que são necessárias para acessar uma visão. Os autores utilizam o mesmo tipo de esquema para quaisquer objetos derivados (índices secundários e múltiplas cópias). Desta forma, o custo de se acessar um índice inclui o custo de sua atualização.

1.3 ACESSO E ALOCAÇÃO DE ARQUIVOS EM BANCOS DE DADOS DISTRIBUÍDOS.

Muitas hipóteses importantes com relação a aspectos de confiabilidade de dados de um banco de dados não são considerados nesta tese, por fazerem parte de políticas adotadas por um SGBD em si. Exemplos deste tipo de hipótese são as utilizadas para análise de algoritmos de controle de concorrência e recuperação (vide , por exemplo, [BER 1984] OU [COP 1988]). Estes e outros trabalhos dicutem como uma queda em um nó pode ser detectada e como proceder à recuperação.

Outros problemas se encaixam neste contexto, como manutenção de log replicado ou não em cada nó, ou política de resolução de conflitos no acesso a dados. Nenhum destes tópicos, embora de

importância na análise de integridade em banco de dados distribuídos, será abordado nesta tese. A razão, como se verá nos capítulos 2 e 3, é que o sistema aqui proposto pressupõe que toda a parte de serialização de transações, procedimento de recuperação de nós, manutenção de múltiplas cópias e envio de mensagens fica a cargo do SGBD. Estes estudos são citados aqui para indicar que não se pretende atacar, com o sistema proposto, o problema mais geral de distribuição e alocação de dados e execução de transações em sistemas distribuídos.

O tratamento de múltiplas cópias de relação é discutido por DANIELS E SPECTOR [DAN 1983], que apresentam um esquema para diretórios duplicados que permite operações concorrentes e disponibilidade de dados e informalmente desenvolvem a proposta da estratégia de duplicação. O algoritmo apresentado utiliza noções de diferimento de atualizações.

O algoritmo apresenta um esquema para diretórios duplicados que permite operações concorrentes e uma alta disponibilidade de dados. A semântica de diretórios duplicados é tipicamente a de diretórios que estão armazenados em um único nó.

A estratégia de duplicação está baseada na manutenção de cópias primárias e secundárias. A cópia primária recebe todas as atualizações, que posteriormente são transmitidas para as cópias secundárias. Uma consulta pode ser enviada a uma cópia secundária, porém a resposta pode não refletir o estado mais recente dos dados. Assim, cada cópia corresponde a uma versão da cópia primária.

O algoritmo associa um número de versão a cada possível chave de cada cópia. Essa técnica permite operações concorrentes sobre diferentes entradas de uma mesma cópia e soluciona certos problemas de implementação da operação de eliminação.

A estratégia de atualização "unânime" consiste em que qualquer operação de atualização deve ser feita em todas as cópias, porém leituras podem ser feitas em qualquer cópia. Essa estratégia de duplicação garante consistência de dados se cada sistema de armazenamento de cópia garantir a consistência de dados localmente.

O algoritmo apresentado em [DAN 1983], segue a filosofia dos algoritmos do "peso do voto". Este tipo de procedimento atribui um certo número (quorum) de votos e um número de versão para cada representante (ou cópia) pertencente a um conjunto de arquivos duplicados. Os tamanhos dos quoruns de leitura e escrita são escolhidos de forma que todo quorum de leitura possua uma intersecção não nula com todo quorum de escrita, o que garante que cada consulta acesse pelo menos uma cópia atualizada de dados.

A estratégia do "peso do voto" possui vários atributos que a tornam particularmente interessante para a base de projetos de diretórios duplicados. Em primeiro lugar, os tamanhos dos quoruns de leitura e escrita podem ser variados de forma a ajustar o custo relativo e a disponibilidade das operações de escrita e leitura. Além disto, consistência e recuperação são principalmente de responsabilidade de transações do sistema de armazenamento que se encarregam de gerenciar a manutenção de cada representante. Como operações concorrentes são sincronizadas através de transações do sistema de armazenamento de cada representante, poderá haver considerável flexibilidade na especificação e implementação do controle de concorrência.

Enquanto os artigos anteriores se referem à manutenção de várias cópias de arquivos de dados, COOPER [COO 1984] descreve um mecanismo para se conseguir maior disponibilidade de programas distribuidos. Em outras palavras, mantém várias cópias de um

mesmo módulo ao mesmo tempo. O algoritmo é uma combinação de chamadas de procedimentos remotos com módulos de programas replicados, visando tolerância a falhas.

Um conjunto de cópias de um módulo é chamado de "troupe". Quando um programa é construido segundo troupes, que aparecem ao usuário como um único módulo, uma chamada de procedimento resulta numa chamada de procedimento replicado (chama várias cópias). Um programa escrito desse modo irá funcionar enquanto pelo menos um membro da troupe "viva" (permaneça operante).

Módulos podem ser replicados em qualquer número de vezes. O sistema admite chamadas de procedimentos remotos, o que permite que um programa tenha módulos localizados em máquinas diferentes. Permite também ao programador escrever programas distribuidos dentro do mesmo padrão utilizado para programas convencionais de computadores centralizados.

A idéia da duplicação tem como objetivo mascarar falhas de componentes individuais. O custo do aumento da confiabilidade de um programa distribuido por duplicação pode ser compensado pelo custo de prover facilidades de recuperação de queda baseada em armazenamentos permanentes tais como "checkpoint" e log de mensagens.

Com o avanço da técnologia, o problema de distribuição e alocação de arquivos deixou de ser algo voltado apenas para aumentar a disponibilidade. Em [COP 1988], por exemplo, os autores descrevem como alocar dados no sistema BUBBA, que é um sistema com alto grau de paralelismo para aplicações que façam uso intenso da dados. Conforme explicam, por alto paralelismo querem dizer que o balanceamento de carga é crítico para o desempenho do sistema; uso intenso de dados (data-intensive) significa que a quantidade de dados é tão grande que as operações

devem ser executadas no local onde estão os dados. Em sistemas deste tipo, é mais eficiente enviar dados intermediários entre nós do que os dados base, tendo em vista o volume dos mesmos.

Este mesmo tipo de problema é discutido por [ABB 1988], que discute os compromissos entre permitir autonomia (de execução) em nós versus transparência na manutenção de cópias de objetos.

Como afirmam os autores, " A pesquisa sobre SGBD`s distribuídos tem se concentrado no problema de acesso aos dados (tradução e decomposição de uma atualização ou consulta em atualizações ou recuperação de dados sobre um conjunto de nós que se comunicam)." Eles observam que ainda há poucos estudos sobre o problema de administração de dados , que é complicada pela existência de fragmentação, cópia e alocação de dados em uma rede. O trabalho de [ABB 1988] discute regras para alocação de fragmentos e cópias de relação, no contexto de um sistema específico (ADAPLEX). Os autores observam que o desejo de autonomia de um nó pode conflitar com o desejo de manutenção de cópias de arquivos em vários nós (já que neste caso a autonomia não pode ser absoluta).

Um estudo recente de comparação de vários algorítmos de controle de concorrência em banco de dados distribuído é feito em [CAR 1988], considerando estratégias de alocação e replicação.

Os autores discutem o fato de que há ainda muito a fazer com relação ao estudo comparativo de desempenho de algoritmos de controle de concorrência em banco de dados distribuídos. Neste sentido, examinam quatro famílias de algoritmos tendo em vista, entre outros, vários níveis de distribuição de dados e de replicação.

Como afirmam, distribuição e cópia de dados podem aumentar o desempenho por permitir execução paralela de consultas e

balanceamento de carga, além de maior acesso aos dados. Por outro lado, tem o inconveniente de criar problemas para controle de concorrência e recuperação. O trabalho dos autores, do ponto de vista de arquivos replicados, difere dos demais, no sentido de que permite diferentes níveis de replicação. Vale entanto, observar que, no que toca a manutenção de cópias de arquivos, os autores comprovaram que quanto maior o número de cópias pior o desempenho devido ao custo de manutenção de atualização. O modelo adotado para esta parte da experiência foi o mesmo que serve de pressuposto para esta tese, como se verá nos capítulos a seguir: supõe-se que atualizações a uma determinada cópia sejam posteriormente enviadas às demais e que transações sejam processadas sequencialmente e localmente. Em outras palavras, conforme afirmam os autores "para casos de cópia única, o sistema age como se houvesse processamento centralizado; para múltiplas cópias, supõe-se que a presença de várias cópias sirva apenas para aumentar a disponibilidade dos dados".

1.4 ESTRUTURA DA TESE.

Os casos da literatura descritos neste capítulo correspondem a alguns dos principais enfoques na pesquisa de pré-processamento e diferimento de atualizações e consultas em banco de dados. No contexto de distribuição, nota-se a preocupação em minimizar tráfego de dados quer através de diferimento, quer de sua alocação. Uma outra situação na literatura é aquela que trata do processamento de atualização em arquivos replicados (diretórios, relações e instantâneos). Neste caso, transações são pré-

processadas para garantir a integridade e consistência dos dados duplicados. Finalmente, observa-se que na atualização de instantâneos existe o cuidado em se evitar processamento de atualizações desnecessárias.

Como veremos adiante, a técnica proposta nesta tese para pré-processamento de atualizações requer a manutenção de algumas estruturas de dados na memória principal. Ela parte do pressuposto de que os dados podem estar distribuídos, quer em fragmentos, quer replicados, mas que existe um nó central que processa transações. É neste nó que é feito o diferimento e o pré-processamento. O sistema utiliza algumas das sugestões de Celis [CEL 1984] e implementa um log condensado semelhante ao descrito por [KAH 1987]. A utilização do sistema para processamento de "refresh" aproveita resultados teóricos de [BLA 1987].

Este capítulo descreveu alguns dos enfoques de pré-processamento e diferimento de atualizações. Os demais capítulos da tese estão organizados da forma descrita a seguir. No capítulo 2 serão apresentadas as definições necessárias ao entendimento desta tese. No capítulo 3 é apresentado o sistema desenvolvido para pré-processar atualizações. O capítulo 4 propõe uma extensão do sistema para processamento de "refresh" de instantâneos. O capítulo 5 discute os testes efetuados utilizando o protótipo implementado e apresenta alternativas para a implementação adotada. O capítulo 6 contém conclusões e propostas para extensão deste trabalho.

2. DEFINIÇÕES.

Este capítulo contém um conjunto de definições necessárias ao entendimento do restante da tese. Supõe-se que o leitor conheça a operações de projeção, seleção e junção natural da álgebra relacional, como, por exemplo, descritas em [MAI 1983] ou [DAT 1986].

2.1 RELAÇÕES E ESQUEMAS

Um **ESQUEMA Ri** é um conjunto de nomes de atributos {Aij}, onde cada atributo corresponde a um domínio de valores possíveis. Um **ESQUEMA RELACIONAL DE BANCO DE DADOS** é um conjunto de esquemas, R={Ri}. Uma **TUPLA** definida sobre o esquema Ri é uma função que para cada atributo em Ri faz corresponder um valor no domínio de atributos. Uma **RELAÇÃO** ri sobre Ri é um conjunto finito de tuplas definido sobre Ri. Um **ESTADO** de um esquema de banco de dados é uma função que para todo o esquema Ri fornece uma relação ri sobre Ri. O conjunto de restrições de integridade I para um esquema relacional R define um estado legal para o banco de dados: um estado é legal se todas as relações no banco de dados satisfazem toda restrição no conjunto I. Uma **OCORRÊNCIA** ou **EXTENSÃO**, de um banco de dados é um conjunto de relações em um estado legal do esquema de banco de dados.

2.2 DEPENDÊNCIA DE DADOS.

As restrições de integridade mais frequentemente mencionadas nesta tese são dependências funcionais. Seja t[A] o indicador da projeção sobre o atributo A da tupla t em álgebra relacional. Uma relação composta pelo conjunto de tuplas {t} satisfaz uma dependência funcional X->A se $\forall$ t1,t2 $\in${t}, t1[X]=t2[X] => t1[A]=t2[A].

Dado um conjunto de dependências funcionais F, existem axiomas de inferência que podem ser usados para inferir todas as dependências funcionais implicadas por F. O **FECHAMENTO** de F, denotado por F+, é o menor conjunto contendo F de modo que a aplicação desses axiomas não possa produzir qualquer dependência funcional fora do conjunto. Nesta tese, se um esquema é sujeito a um conjunto de dependências funcionais, esse conjunto será considerado um conjunto não redundante, onde todas as dependências são reduzidas à esquerda. Uma dependência X->Y em F+ é **reduzida à esquerda** se X não contém nenhum subconjunto X' tal que X'-> Y esteja em F+. Um conjunto de dependências funcionais é **NÃO-REDUNDANTE** se F não contém um subconjunto próprio F' onde F'+ = F+; ou seja F $\equiv$F'.

Esquemas são muitas vezes **NORMALIZADOS**, isto é, decompostos por projeções especiais, para eliminar alguns tipos de redundâncias e solucionar certos problemas de atualização. A única forma normal referenciada nesta tese é a **FORMA NORMAL BOYCE CODD (BCNF)**. Um esquema Ri está em BCNF se todo o lado esquerdo de uma dependência não trivial em um esquema determina totalmente o esquema. Dependências triviais são aquelas do tipo X -> X. Quando o esquema está em BCNF diz-se também que obedece a

dependências (funcionais) **de chave**. Uma dependência de chave é aquela em que o lado esquerdo constitui uma chave do esquema. Seja R um esquema sujeito a um conjunto F de dependências de chave e sejam C1,C2,...,CN chaves distintas de R. Então, F+ = {C1 -> R, C2 -> R ... CN -> R} é não redundante. Neste caso, as dependências são indicadas pela enumeração das chaves, não sendo necessário especificar seus lados direitos). Em resumo, havendo apenas dependências funcionais, e estando o esquema em BCNF, a descrição de suas restrições de integridade é feita pela enumeração das chaves. Em geral apenas uma chave é designada. Na prática, ela é chamada de chave primária.

O motivo pelo qual a BCNF foi escolhida como ponto básico nesta tese é porque diminui a complexidade dos testes de integridade que precedem a atualização (restringem-se a verificação de chave). A maioria dos estudos na área de atualização de banco de dados (por exemplo [DAY 1982],[KEL 1986] parte do princípio que as relações devem estar em BCNF. O tratamento das atualizações feito na tese pode ser estendido para 3NF, embora isto acarrete um maior número de testes e impeça algumas das verificações de integridade efetuadas quando da inserção ou modificação de tuplas.

2.3 SISTEMA DO GERENCIAMENTO DO BANCO DE DADOS.

Um sistema de banco de dados envolve quatro componentes maiores: dados, **hardware**, **software** e usuários.

Entre o banco de dados físico (os dados armazenados) e os usuários do sistema encontra-se uma camada de **software**,

geralmente denominada de sistema de gerenciamento de banco de dados ou **SGBD**. Todas as solicitações dos usuários para acesso ao banco de dados são tratadas pelo SGBD. No geral, a função do SGBD é isolar os usuários do banco de dados dos níveis mais baixos de detalhes, de **hardware**. Assim, o SGBD fornece uma visão do banco de dados acima do nível de **hardware**, e suporta a operação do usuário expressa em termos daquela visão em nível mais alto.

2.4 NÍVEIS DE DEFINIÇÃO DE UM BANCO DE DADOS.

Pelo modelo [ANS 1975], a definição de um banco de dados deve ser feita segundo três níveis gerais: físico (interno), lógico e externo.

(1) **NÍVEL INTERNO** - É o nível que está mais próximo do armazenamento físico, ou seja, está voltado para a forma como os dados estão realmente armazenados;

(2) **NÍVEL EXTERNO**- É o nível mais próximo dos usuários, isto é, o que está voltado para a forma como os dados são vistos por cada usuário. Muitas vezes é chamado de visão do usuário;

(3) **NÍVEL LÓGICO**- É um nível situado entre os dois anteriores, onde se define o projeto lógico do banco de dados. As visões do nível externo são integradas em um único nível conceitual lógico.

Como já foi dito anteriormente, o nível externo está voltado para as visões de cada usuário. Desse modo o nível lógico pode ser imaginado como definindo a visão da comunidade de usuários. Isto é, haverá muitas "visões externas", cada uma consistindo em

uma representação abstrata de parte do banco de dados (a maioria dos usuários está interessada numa porção restrita do banco de dados). Da mesma forma, haverá uma única "visão interna", representando o banco de dados como ele está na realidade armazenado. As operações no nível externo devem ser mapeadas para o nível lógico e deste para o físico. Operações a nível físico são refletidas no nível externo através do mapeamento inverso.

Esta tese se aterá a considerações referentes ao modelo lógico de banco de dados (nível conceitual), além de implementar operações de atualização em função de solicitações a nível externo. Estas solicitações serão traduzidas em operações de atualização sobre o modelo lógico, sob a forma de relações. Cabe ao SGBD utilizado fazer o mapeamento final para o modelo físico.

2.5 ATUALIZAÇÕES EM UM BANCO DE DADOS.

A tese desenvolve o pré-processamento de solicitações de atualizações de tuplas de relações. Convém observar que é vantajoso, por exemplo, pré-processar sequências de inserções e eliminações porque desse modo evitamos inserir tuplas que posteriormente serão eliminadas e consequentemente ocorre a otimização de acessos a nós onde residem as relações envolvidas.

Vamos considerar agora atualizações de relações em BCNF. Sabemos pela sua definição que nenhum atributo depende transitivamente de uma chave. Portanto, podemos concluir que qualquer atributo que determina funcionalmente outro atributo é uma chave. Assim, para efeitos de inserção, eliminação e alteração de relações em BCNF, temos que nos preocupar apenas com

as chaves. Basta informar ao SGBD sobre as restrições de integridade indicando os atributo(s) constituinte(s) da chave. Todos os outros atributos são funcionalmente dependentes desse atributo ou combinação de atributos, sendo esta restrição mantida pelo SGBD.

Este capítulo apresentou conceitos formais necessários ao entendimento da tese. O capítulo seguinte trata do pré-processamento de atualizações.

Para efeitos de atualização esta tese irá considerar as seguintes operações sobre tuplas de relações:

1-**INSERÇÃO**: inserir dados referentes a uma nova chave, isto é, inserir uma nova tupla no banco de dados.

2-**ELIMINAÇÃO**: dada uma chave, eliminar a tupla correspondente do banco de dados.

3-**ALTERAÇÃO**: dada uma chave alterar a tupla, isto é, mudar o conteúdo do campo ou campos da tupla correspondente do banco de dados (exceto a chave).

Algumas das hipóteses aqui feitas também são adotadas em [ABB88]:

- os usuários enxergam uma visão logicamente centralizada do dicionário de dados e do banco de dados, sendo que a alocação de dados a nós lhes é transparente, bem como a existência de múltiplas cópias.

-se o usuário acessa uma relação fragmentada o SGBD automáticamente decompõe as consultas em consultas aos fragmentos.

3.SISTEMA DE PRÉ-PROCESSAMENTO DE ATUALIZAÇÕES.

Este capítulo apresenta uma abordagem para solução de pré-processamento de atualizações, baseada em utilização de logs condensados. O sistema discutido pode ser usado também na manutenção de instantâneos, visões materializadas e versões de arquivos.

3.1 HIPÓTESE DO AMBIENTE DE TRABALHO.

Esta tese pressupõe a existência de um banco de dados distribuído onde relações (ou fragmentos) a serem atualizadas podem ou não estar replicadas. Os pedidos de atualização são centralizados e pré-processados pelo sistema aqui proposto e depois enviados aos nós correspondentes. Este sistema tem como entrada comandos de atualização do usuário, transmitidos pelo SGBD, e como saída transações com comandos de atualização para o SGBD (já condensados).

O sistema não se preocupará com problemas referentes a arquitetura da rede, nem com serialização, integridade e recuperação de transações em sistemas distribuídos. Supõe-se igualmente que o SGBD já haja feito a consistência de dados, no que diz respeito a domínio de atributos. Desta forma, o sistema proposto se aterá apenas ao pré-processamento e diferimento de atualizações como se efetuado em um nó que centraliza o processa-

mento das atualizações do banco de dados. Este nó será chamado de **nó de processamento** para distinguí-lo dos demais. Note que esta hipótese permite tratar de forma indistinta sistemas onde há replicação de relações e onde isto não ocorre. Basta haver um diretório central no próprio nó de processamento indicando quais relações estão replicadas em cada nó. As atualizações para uma relação serão automaticamente enviadas para as suas cópias após o pré-processamento, sendo que o SGBD se encarrega de controlar este envio.

O sistema de pré-processamento recebe as solicitações de atualização (que podem vir de qualquer nó da rede), efetua o seu pré-processamento e gera transações para sua execução pelo SGBD. O controle do recebimento das solicitações dos usuários, bem como da execução das transações, é deixado aos controles apropriados do SGBD, que também se encarrega de transmitir aos usuários, quando cabível, as mensagens emitidas pelo sistema de pré-processamento (embora o protótipo implementado possua interface direta com o usuário). Do ponto de vista deste sistema, tudo funciona como se tratasse de um banco de dados centralizado, uma vez que os controles característicos de banco de dados disitribuídos são transparentes ao sistema. Por exemplo, se forem geradas várias transações de atualização para cópias de uma mesma relação e alguns dos nós não estiverem ativos, caberá ao SGBD distribuído gerenciar a execução posterior destas transações.

Conforme se verá adiante, a geração das transações não é imediata, já que o sistema de pré-processamento executa também mecanismos de diferimento das operações. Em outras palavras, o ambiente de trabalho pressuposto é semelhante ao descrito no capítulo 1 em trabalhos como os de [BLA 1986, CEL 1984, KAH 1987]. A manutenção da consistência dos nós onde possívelmente

haja relações replicadas pode ser feita, por exemplo, através de trabalhos como os mencionados no mesmo capítulo [EAR 1983, DAN 1984].

Se R é uma relação do banco de dados relacional, sua verificação de integridade será feita por chaves, e através das dependências funcionais da relação R, que deve estar normalizada em BCNF.

Outras hipóteses do ambiente de trabalho já foram definidas no capítulo 2.

3.2 A ESTRATÉGIA DO PRÉ-PROCESSAMENTO DE ATUALIZAÇÕES.

Pelo método tradicional de atualizações, ao menos dois acessos são necessários a cada atualização de tupla:

(1) Acesso no momento que a requisição foi recebida (que pode requerer mais de um acesso físico, dependendo dos índices, do tamanho e caracteristícas da relação a ser atualizada). No caso de inserção de tupla, é necessário verificar se a tupla já existe, caso em que a inserção não é permitida.

(2) Armazenamento da relação atualizada (o que novamente pode requerer mais de um acesso).

Desse modo, se um pedido de atualização seleciona uma considerável quantidade de tuplas, essa operação torna-se muito cara em termos de acesso a disco (ou número de vezes que um determinado nó deve ser acessado). Se essas operações forem pré-processadas, é possível que haja diminuição de tráfego na rede . Na verdade, esta justificativa para pré-processamento de atualizações é semelhante à utilizada para otimização de

consultas.

Note que no caso de um pedido de eliminação posterior ao pedido de inserção diferida de uma determinada tupla, não haverá necessidade de acessar o nó onde reside a relação à qual a tupla pertence. O pré-processamento permite, igualmente, uma concentração de tráfego, ou seja, permite otimização na abertura e fechamento de arquivos.

Para registrarmos o fato que determinadas tuplas devem ser atualizadas precisamos saber:

(1) Quais tuplas a serem afetadas (chave e relação afetada);

(2) Qual operação a ser aplicada;

(3) Se a tupla já tem pedido anterior de atualização pendente.

O sistema proposto utiliza um conjunto de estruturas de dados para reter estas e outras informações sobre as atualizações. O método que esta tese propõe para armazenar essas informações é o de associar as solicitações de usuários a um "**Log de atualizações**". Para facilitar a verificação de (1) e (3), além do log, é mantido uma estrutura segundo uma árvore B [COM 1979] contendo todas as chaves de tuplas existentes na relação R a ser atualizada. Para cada valor de chave, haverá indicação da existência de atualizações pendentes, para a tupla correspondente.

É mantido um par (árvore-B,log) para cada relação passível de atualização. O número de cópias de uma relação é irrelevante: se houver duplicação, será mantida uma árvore por relação e as atualizações serão enviadas pelo SGBD, após o pré-processamento, a todos os nós que possuirem cópias. A fragmentação de relações é transparente ao sistema.

O sistema de pré-processamento de atualizações sugerido consiste em três módulos. O primeiro, de inicialização, gera uma estrutura a partir das chaves das relações passíveis de atualização ; o segundo módulo processa e condensa na estrutura as solicitações de atualizações que chegam ao nó de processamento. O sistema de pré-processamento necessita das seguintes informações:

a) Tipo de operação (inserção, modificação ou eliminação);

b) Nome da relação a atualizar;

c) Nome e o valor do atributo chave da tupla;

d) Nomes e os valores dos demais atributos, se necessário (para inserção / ou modificação de tuplas).

O sistema verifica se o pedido é válido (por exemplo, se a chave não está presente no índice deve se tratar de inserção de uma nova tupla). Se o pedido não for válido, o SGBD é avisado pelo sistema de pré-processamento e por sua vez informa ao usuário. Se o pedido for válido, a operação é processada e condensada. Mais adiante a condensação dessas operações será explicada com maiores detalhes. O terceiro módulo "transforma" posteriormente a estrutura em transações de atualização com comandos para o SGBD . Este último módulo é ativado nas seguintes situações:

a) por solicitação do usuário (por exemplo, se especificado ao final de uma sessão) [CAS 1988];

b) quando a área ocupada pela estrutura ultrapassa os limites definidos pelo administrador do banco de dados;

c) quando há consulta a uma tupla com atualização pendente [CEL 1984], [ROU 1986];

d) ao ser encerrada a execução do sistema. Idealmente, o sistema deve permanecer sempre ativo, embora essa seja uma opção que cabe ao administrador do banco de dados.

As transações geradas pelo terceiro passo são opcionalmente armazenadas em um arquivo executável que pode ser processado posteriormente pelo SGBD. Por exemplo, se as atualizações se referem apenas a relações em um nó desconectado temporáriamente da rede, o arquivo pode ser enviado para execução no nó quando for recuperada a comunicação. O nó é então acessado e efetuadas as atualizações pendentes. Este tipo de procedimento é semelhante ao utilizado em "refresh" de instantâneos replicados.

Como já vimos anteriormente, as únicas restrições analisadas são dependências funcionais, sendo que as relações devem estar em BCNF. Desta forma, tanto atualizações como verificação de integridade são feitas a partir dos valores das chaves.

As operações permitidas são:

(1) Inserir tupla,

(2) Eliminar tupla,

(3) Alterar campo não chave da tupla,

(4) Consultar tupla.

A consulta pode ser respondida a partir das estruturas de dados (sem necessidade de acesso à relação) nos seguintes casos:

a) tupla foi eliminada,

b) tupla inexistente e

c) tupla foi inserida e a atualização ainda não processada.

Caso haja consulta a tupla com atualização (ou condensação de atualizações) pendente, o sistema prevê execução da atualização, gerando o comando correspondente para o SGBD.

Para uma dada relação, o sistema de pré-processamento funciona da seguinte forma:

a) Módulo de inicialização: cria a árvore índice (árvore B) para as chaves da relação, indicando que nenhuma tupla foi atualizada e aloca espaço para o log;

b) Módulo de pré-processamento: armazena solicitações de atualização no log; se uma tupla já tem indicação de atualização pendente (o que pode ser verificado através da árvore índice), modifica o log de modo a retratar o conjunto (condensado) de atualizações solicitadas. Atualiza entradas do índice (árvore B) quando cabível.

c) Módulo gerador de transações: a partir do log de atualizações, gera transações de atualizações em linguagem inteligível pelo SGBD. Libera as entradas correspondentes no log e modifica a árvore índice adequadamente.

Esta arquitetura do sistema facilita sua adaptação a vários SGBDs, uma vez que apenas o terceiro módulo tem grande dependência do SGBD. O primeiro módulo utiliza comandos do SGBD apenas para consultar esquemas e percorrer relações. O segundo módulo não utiliza comandos DML (linguagem de manipulação de dados).

3.3 MANUTENÇÃO DO LOG E DO ÍNDICE DE CHAVES.

3.3.1 A ESTRUTURA DE DADOS DO LOG DE ATUALIZAÇÃO.

Como já dissemos anteriormente, temos que saber quais tuplas devem ser atualizadas e qual modificação deve ser aplicada. Esta informação é guardada, para cada tupla, no **log de atualizações**, formado por uma lista encadeada.

Cada registro do log tem o seguinte conteúdo:

* A chave da tupla,

* A operação a ser efetuada (códigos correspondentes às operações de eliminação, inserção ou alteração)

* O conteúdo da tupla ou campos da atualização (dependendo da operação), sob forma de cadeia de atributos a serem atualizados.

Registros do log são atualizados durante o processo de condensação de atualizações: por exemplo, uma inserção seguida de uma remoção insere e elimina um registro do log.

A sugestão desta tese é manter o log como uma lista circular duplamente ligada com cabeça de lista. Esta representação foi escolhida para facilitar a manutenção dinâmica do log. O acesso a um registro é feito através do índice de chaves, descrito a seguir, que contém apontadores para cada registro do log. O sistema dispõe também de um "depósito" de registros livres denominado de **lista livre**. Ocorrendo um pedido de atualização, é retirado um registro da lista livre, e à medida em que o pré-processamento é efetuado os registros são devolvidos novamente para a lista livre. Quando as transações são geradas, todos os registros correspondentes do log devem ser devolvidos para a lista livre.

A detecção de quando o log está cheio, sinal de que todas as transações devem ser geradas, é feita observando a lista livre, ou seja, quando a lista livre não contiver mais nenhum registro disponível. Em outras palavras, se um novo pedido de atualização não pode ser incorporado ao log, todas as transações devem ser geradas, processando inclusive a última atualização. A cabeça de lista é utilizada para detectar quando o log está vazio, ou seja, não existem atualizações pendentes.

As operações permitidas sobre o log de atualizações são: inserção (novos pedidos de atualizações a tuplas da relação R),

remoção (resultante de uma eliminação posterior a uma inserção de uma mesma tupla ou fim de processamento) e alteração (resultante da condensação de operações).

Finalmente, um registro pode ser retirado do log quando há consulta à tupla correspondente (no caso de modificação). Neste caso, a transação de atualização é gerada. A opção de retirar ou não o registro é deixada ao administrador do banco de dados e corresponde à estratégia preguiçosa de [ROU 1986].

3.3.2 A ESTRUTURA DE DADOS DO ÍNDICE DE CHAVES.

Um índice de chaves deve ser mantido para cada relação, para saber se uma determinada tupla já recebeu pedido anterior de atualização e para verificação, ainda que primitiva, de integridade. O índice também pode ser utilizado para auxiliar a consulta ao banco de dados, já que informa o status de atualização de uma tupla.

A figura a seguir nos mostra o conteúdo de uma entrada do índice.

FIG 3.1: REGISTRO DO ÍNDICE DE CHAVES.

Esse índice está armazenado em uma árvore-B. Cada entrada de tupla do índice possui os seguintes campos:

- a chave CHi da tupla da relação R;
- o código de atualização;
- o apontador para o registro de atualização no log.

Para efeitos de consulta temos os seguintes códigos de atualização:

I: Tupla está no log e ainda não foi inserida

E: Tupla está no log e ainda não foi eliminada

A: Tupla está no log e ainda não foi alterada

NE: Tupla não existe (tupla foi inserida e eliminada)

#: Tupla esta em R e não existe pedido de atualização pendente.

Note que no caso da operação **eliminar**, caso a tupla não exista, é suficiente uma consulta ao índice de chaves para tomar conhecimento do fato: ao procurar a chave, esta não é encontrada ou a tupla já foi eliminada.

O mesmo vale para operação **inserção** caso haja consulta ou tentativa de inserir alguma tupla já existente.

Assim para efeitos de atualização temos:

ELIMINAR- Se a tupla existe, registrar a operação no log. Caso contrário há mensagem de erro,

ALTERAR- Caso a tupla não existir ocorre mensagem de erro, senão registra-se no log a operação.

INSERIR- Se a tupla já existe, haverá mensagem de erro. Se não, registra-se no log a operação.

Portanto nestes casos de erro, não há necessidade de acessar a relação ou gravar no log.

O capítulo 5 apresenta uma discussão da validade de manter-

se todas as chaves no log para permitir esta verificação preliminar de integridade.

Também para consultas ao banco de dados as informações do log e do índice podem ser utilizadas, como, por exemplo, leitura de uma tupla cuja inserção foi solicitada, está presente no log de atualizações mas ainda não foi enviada para a relação correspondente.

3.3.3 PROCEDIMENTO DE MANIPULAÇÃO DO LOG E DO ÍNDICE DE CHAVES.

As tabelas a seguir resumem o efeito de condensar operações no log e no índice, dada a atualização do cabeçalho (primeira operação) seguida pela atualização da coluna (segunda operação).

1a. OPERAÇÃO / 2a. OPERAÇÃO	INSERÇÃO	ELIMINAÇÃO	MODIFICAÇÃO
INSERÇÃO	INVÁLIDA	INSERE NO LOG	INVÁLIDA
REMOÇÃO	REMOVE DO LOG	INVÁLIDA	REMOVE DO LOG
MODIFICAÇÃO	MODIFICA NO LOG	INVÁLIDA	MODIFICA NO LOG

FIG 3.2: CONDENSAÇÃO NO LOG.

1a. OPERAÇÃO / 2a. OPERAÇÃO	INSERÇÃO	ELIMINAÇÃO	MODIFICAÇÃO
INSERÇÃO	INVÁLIDA	I	INVÁLIDA
ELIMINAÇÃO	NE	INVÁLIDA	E
MODIFICAÇÃO	I	INVÁLIDA	A

FIG 3.3: CONDENSAÇÃO NO ÍNDICE.

Como dito anteriormente, as operações permitidas sobre o log de atualizações são: inserção, remoção e alteração. A apêndice mostra os procedimentos para cada uma delas. Na realidade esses procedimentos, além de efetuar as operações citadas, tratam também da interação do log de atualizações com o índice de chaves, o que, como veremos, é nada mais que o pré-processamento iterativo.

Na implementação do protótipo, o conteúdo dos atributos é fornecido pelo usuário. Se o pedido é de inserção, o usuário fornece ao sistema o nome e o valor de cada atributo (estes últimos são passados campo a campo, que são iterativamente colocados em uma estrutura apropriada para inserção no log). Se o pedido é de modificação, o usuário fornece além da chave, o nome e o novo valor do atributo a ser modificado. Se já existir pedido anterior de atualização , essa operação é condensada no registro correspondente no log, caso contrário um novo registro é adicionado ao log de atualizações. Se o pedido for de eliminação, não é necessário passar mais informações ao sistema além daquelas já fornecidas (chave e nome da relação). Uma atualização do tipo inserção tem os atributos armazenados na ordem fornecida pelo esquema.

A descrição destes procedimentos é feita no apêndice, usando a metodologia de definição de tipos abstratos de dados, com terminologia de LISKOV[LIS 1979]. Note que estes procedimentos descrevem o manuseio do log e não atualizações sobre as relações.

Se o usuário acessa dados replicados, o SGBD automaticamente transforma este acesso em acesso a uma dada cópia em um certo nó. Para o caso de atualização, o SGBD gerencia sua propagação para todas as cópias de forma a manter a consistência.

Não serão levadas em conta considerações como critérios de

alocação de dados, cópias e fragmentos ou grau de autonomia de um nó. A este respeito, sugere-se ao leitor o estudo de [COP 1988] ou [ABB 1988].

3.3.4 ESTRUTURA PARA TRATAMENTO DE MULTI-RELAÇÕES.

As estruturas discutidas anteriormente se referem a uma única relação. O tratamento multi-relação, conforme mencionado anteriormente, considera um par (árvore, log) para cada relação passível de atualização. Sugere-se definir uma matriz onde cada entrada corresponde a uma relação e contém três elementos:

* nome da relação;
* ponteiro para a raiz da árvore B correspondente à relação;
* nomes dos atributos da relação (esquema obtido do SGBD).

A figura a seguir mostra a estrutura correspondente:

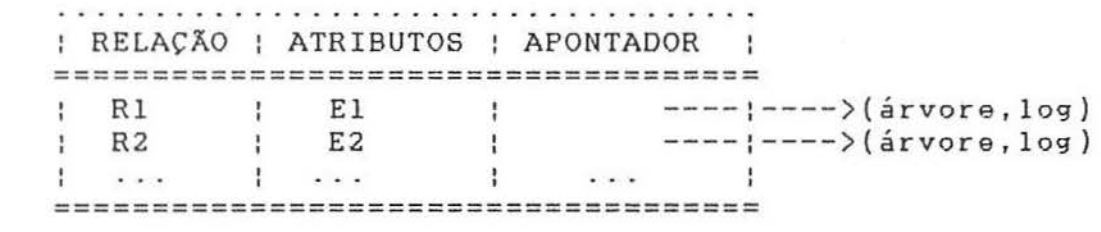

FIG 3.4: ESTRUTURA DOS PARES (ÁRVORE,LOG).

Esta estrutura permite que conjuntos diferentes de relações possam ser processados em ativações diferentes do sistema.

3.3.5 EXEMPLOS.

Esta seção apresenta exemplos do funcionamento do algoritmo de pré-processamento, para uma única relação R.

Inicialmente temos o log de atualizações vazio e o índice de chaves associado a relação R. O campo código de atualização contém "#", o que significa que a tupla está em R e não existe pedido de atualização pendente. Quando um pedido de inserção é efetuado pelo usuário os seguintes passos são tomados:

1) É verificado se o pedido é válido através do índice de chaves (isto é, se a tupla não existe na relação);

2) É inserido então um nó na árvore contendo a chave da nova tupla e o rótulo correspondente à operação inserção (I);

3) É requisitado um registro para o log que conterá o pedido de inserção. O campo apontador no índice de chaves conterá o endereço desse registro. Os atributos são armazenados, em cadeia, na ordem do esquema.

A figura 3.5 mostra como ficam o índice de chaves e o log de atualizações.

O segundo exemplo mostra um pedido de eliminação dessa mesma tupla. O registro do log que contém o pedido de inserção é removido e os campos de atualização e apontador no índice de chaves passam a ter os códigos "NE" (tupla não existe) e "NULL" respectivamente. A figura 3.6 mostra essa situação.

Se um pedido de inserção é efetuado e já existe um pedido anterior de eliminação, o código no índice de chaves é atualizado

com o rótulo "I" (inserção).

O terceiro exemplo de pré-processamento de atualização mostra quando há pedidos de modificação consecutivos de uma tupla. Podemos ter as seguintes situações.

1) Não existe pedido anterior de atualização (ou seja, o campo do código no índice de chaves contém o rótulo "#")

1a) É requisitado um registro livre que conterá o pedido de atualização. Esse registro é inserido no log;

1b) O campo código do índice de chaves é atualizado com o código de atualização correspondente ("A");

1c) O campo apontador no índice terá o endereço desse registro (FIG 3.7);

2) Existe pedido anterior de alteração.

2a) É efetuado um novo pedido de alteração desse mesmo campo dessa mesma tupla.

Ao consultar o índice de chaves é verificado que já existe um pedido anterior. Nesse caso basta através do campo apontador no índice acessar o registro no log e indicar a nova alteração (FIG 3.8)

2b) É efetuado um novo pedido de alteração dessa mesma tupla, porém para um campo diferente do anterior. Do mesmo modo, o registro no log é acessado, e é acrescentado na cadeia de alterações o novo valor para o campo desejado (FIG 3.9).

3) Já existe pedido anterior de inserção. O rótulo de inserção ("I"), permanece no índice de chaves, o registro no log é acessado e a atualização é efetuada para o campo correspondente (FIG 3.10).

O quarto exemplo mostra quando há pedido de eliminação. Podemos ter as seguintes situações:

3.1) Não existe nenhum pedido anterior de atualização. Aqui é tomado o mesmo procedimento que no exemplo 1. O campo código no

índice de chaves é atualizado para "E" (FIG 3.11).

3.2) Existe pedido anterior de inserção.

Nesse caso o registro no log é removido e o rótulo no índice da chaves é atualizado para "NE" (tupla não existe)(FIG 3.12).

3.3) Existe pedido anterior de modificação.

O código no índice de chaves, e o campo correspondente no log são atualizados para "E" (FIG 3.12).

A figura 3.13 mostra as regras de forma resumida adotadas nos exemplos.

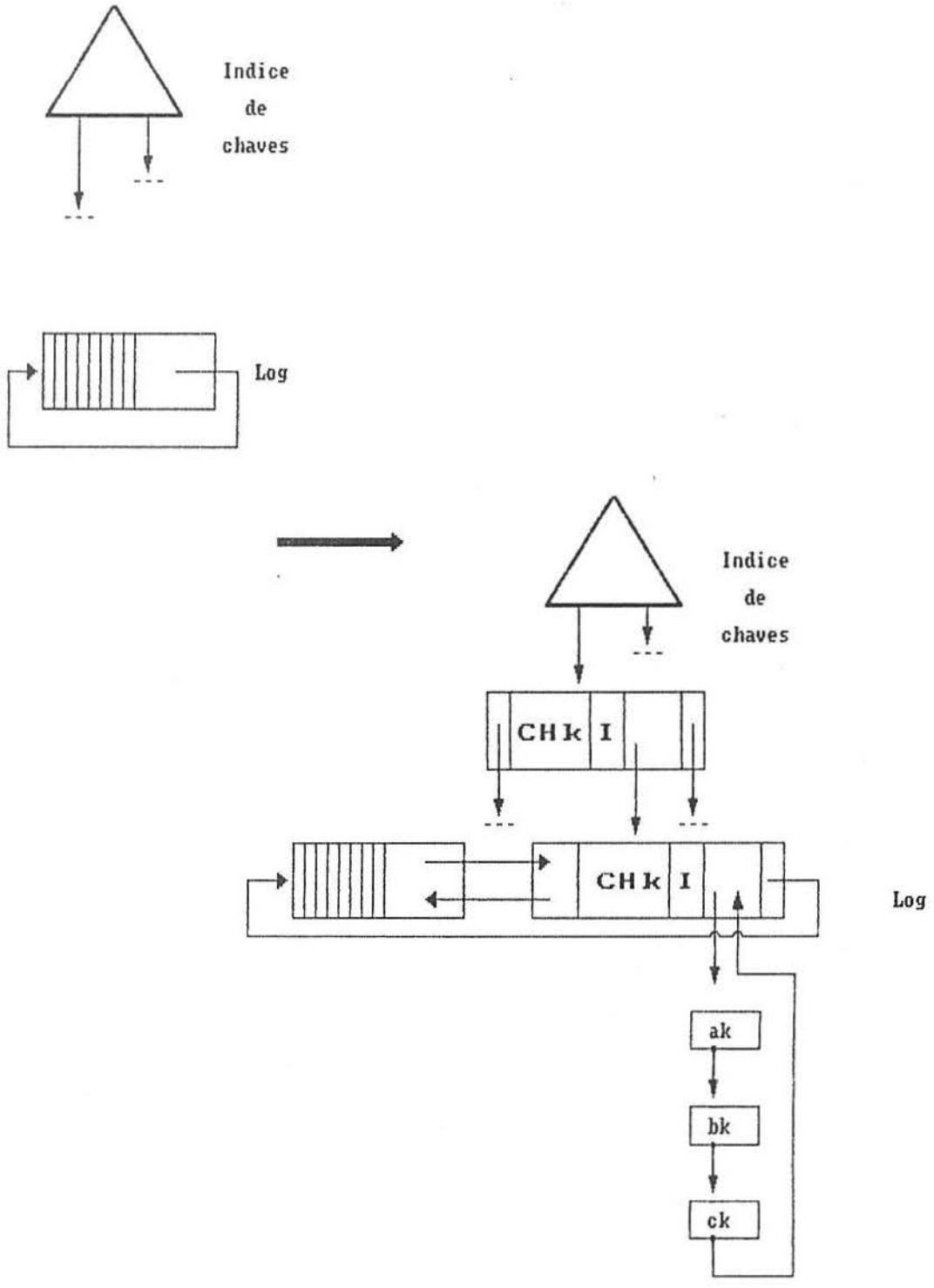

FIG 3_5 : PEDIDO DE INSERCAO SEM PEDIDO ANTERIOR DE ATUALIZACAO. INICIALMENTE TEMOS O LOG DE ATUALIZACOES VAZIO E O INDICE DE CHAVES ASSOCIADO A RELACAO R, E NAO EXISTE PEDIDO DE ATUALIZACAO PENDENTE.

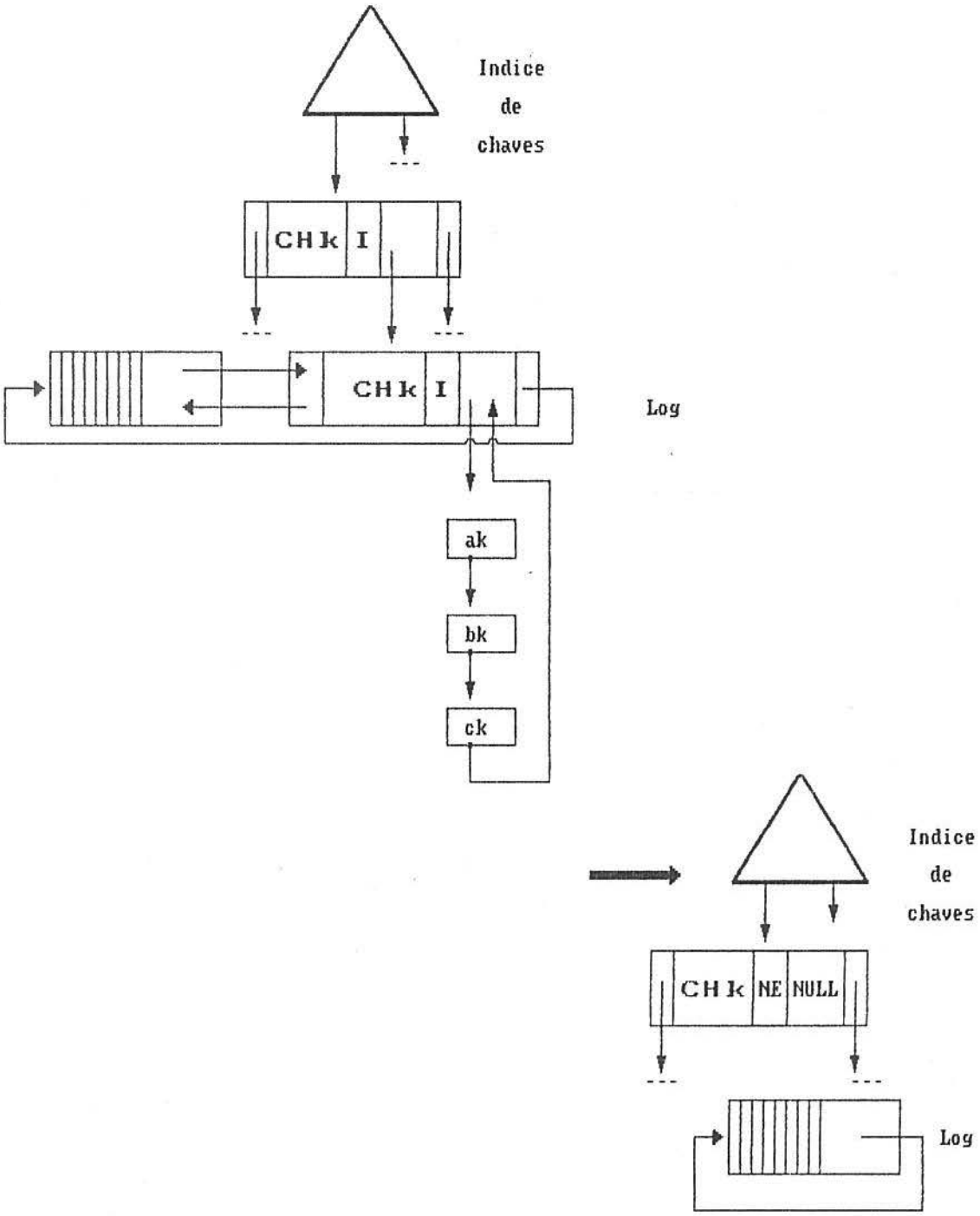

FIG 3.6 : PEDIDO DE ELIMINACAO DE UMA TUPLA COM PEDIDO ANTERIOR DE INSERCAO.

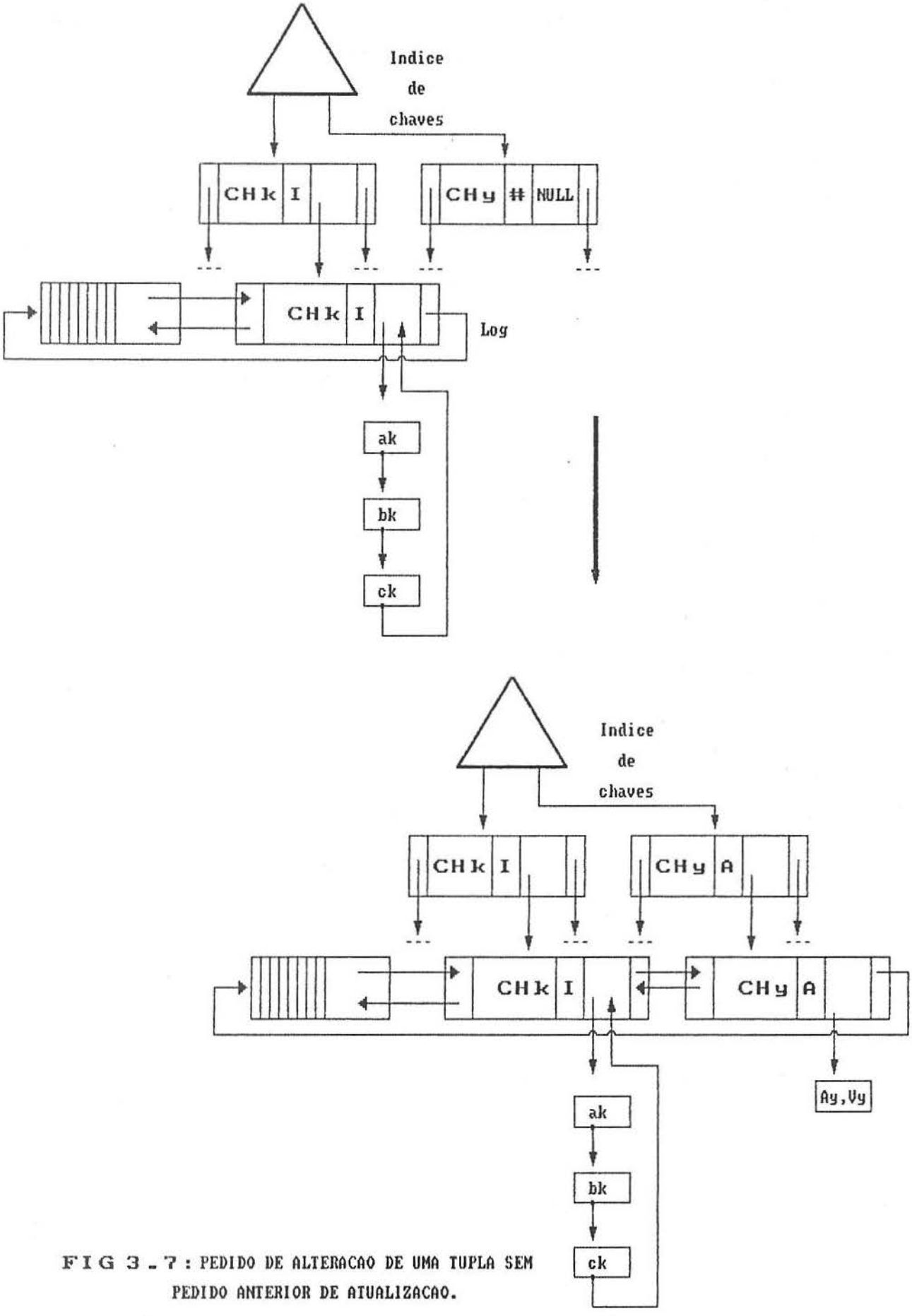

FIG 3_7 : PEDIDO DE ALTERACAO DE UMA TUPLA SEM PEDIDO ANTERIOR DE ATUALIZACAO.

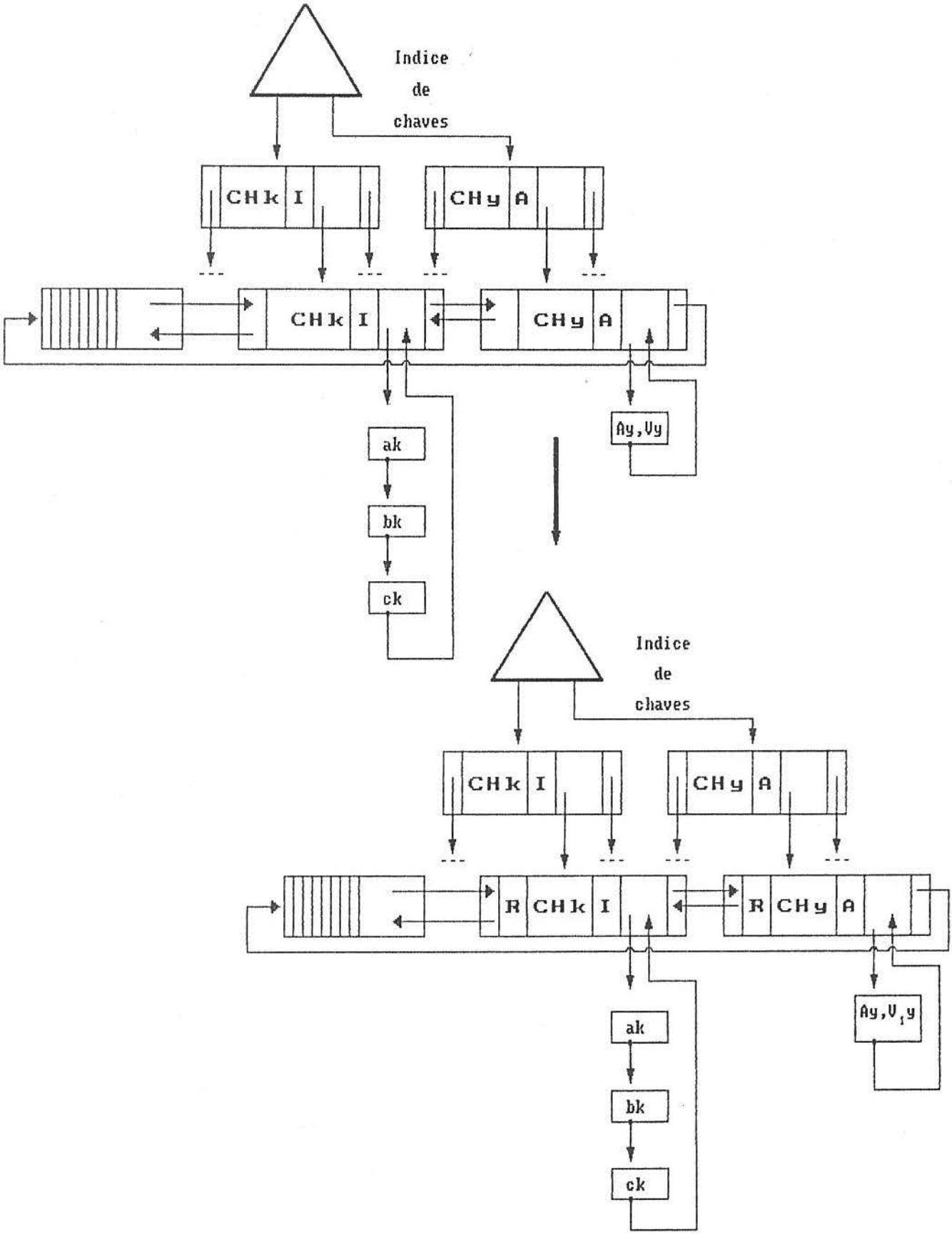

FIG 3.8: PEDIDO DE ALTERACAO DE UMA DETERMINADA TUPLA,

COM PEDIDO ANTERIOR DE ATUALIZACAO.

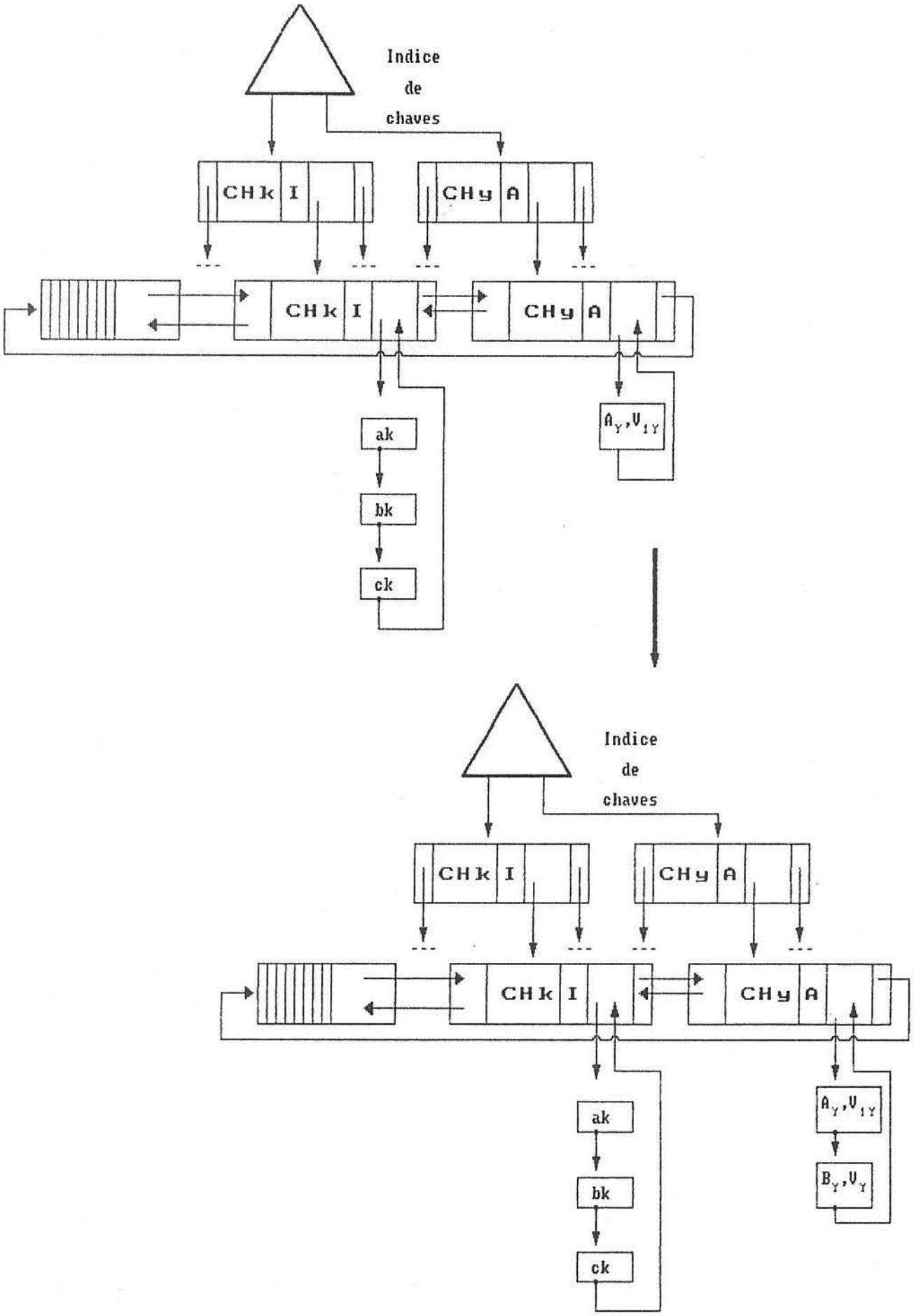

FIG 3.9 : ALTERACAO DE UMA TUPLA COM PEDIDO ANTERIOR DE ATUALIZACAO, PORÉM DE UM CAMPO DIFERENTE DO PEDIDO ANTERIOR.

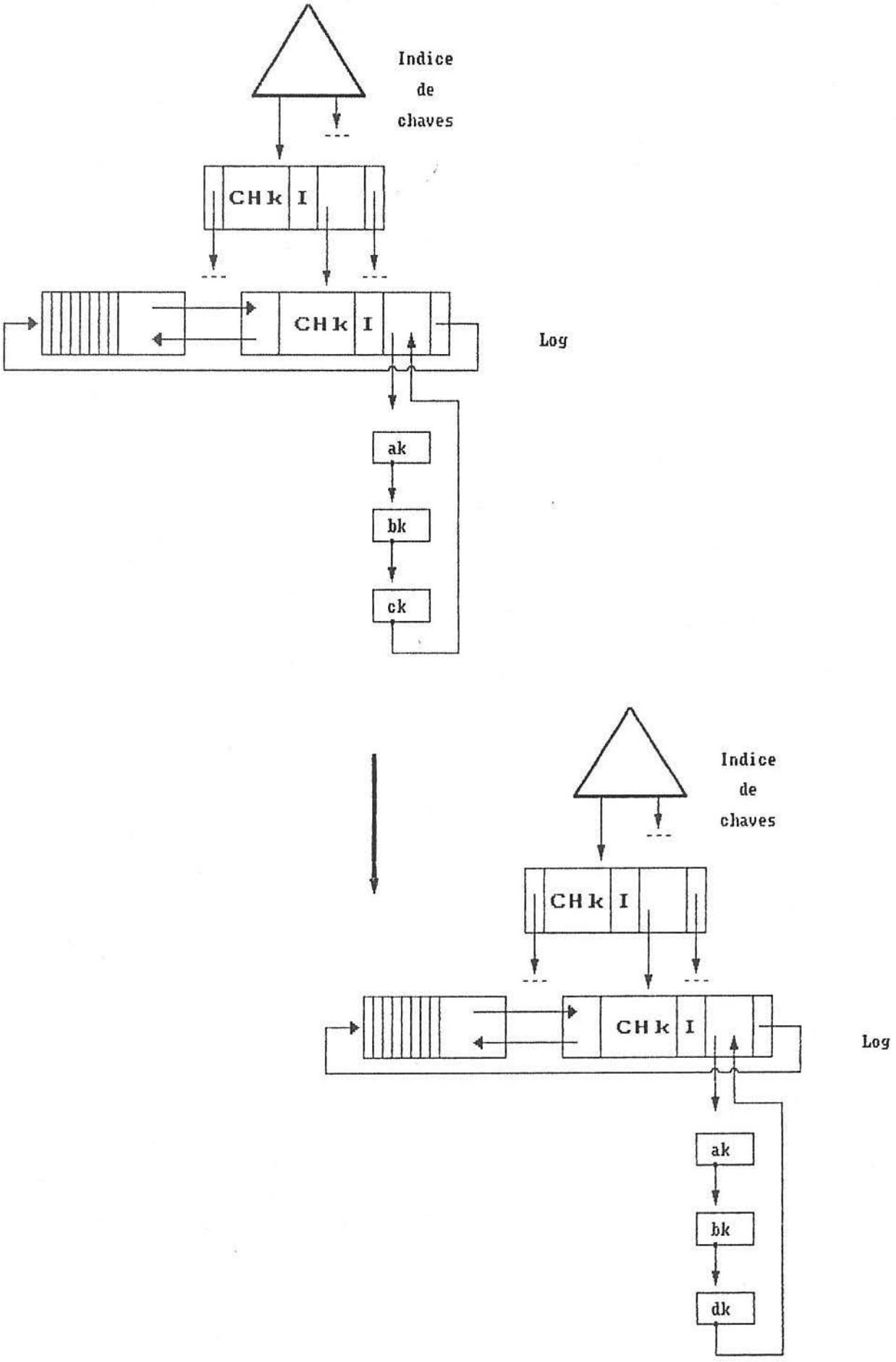

FIG 3.10: PEDIDO DE ALTERACAO COM PEDIDO ANTERIOR DE INSERCAO.

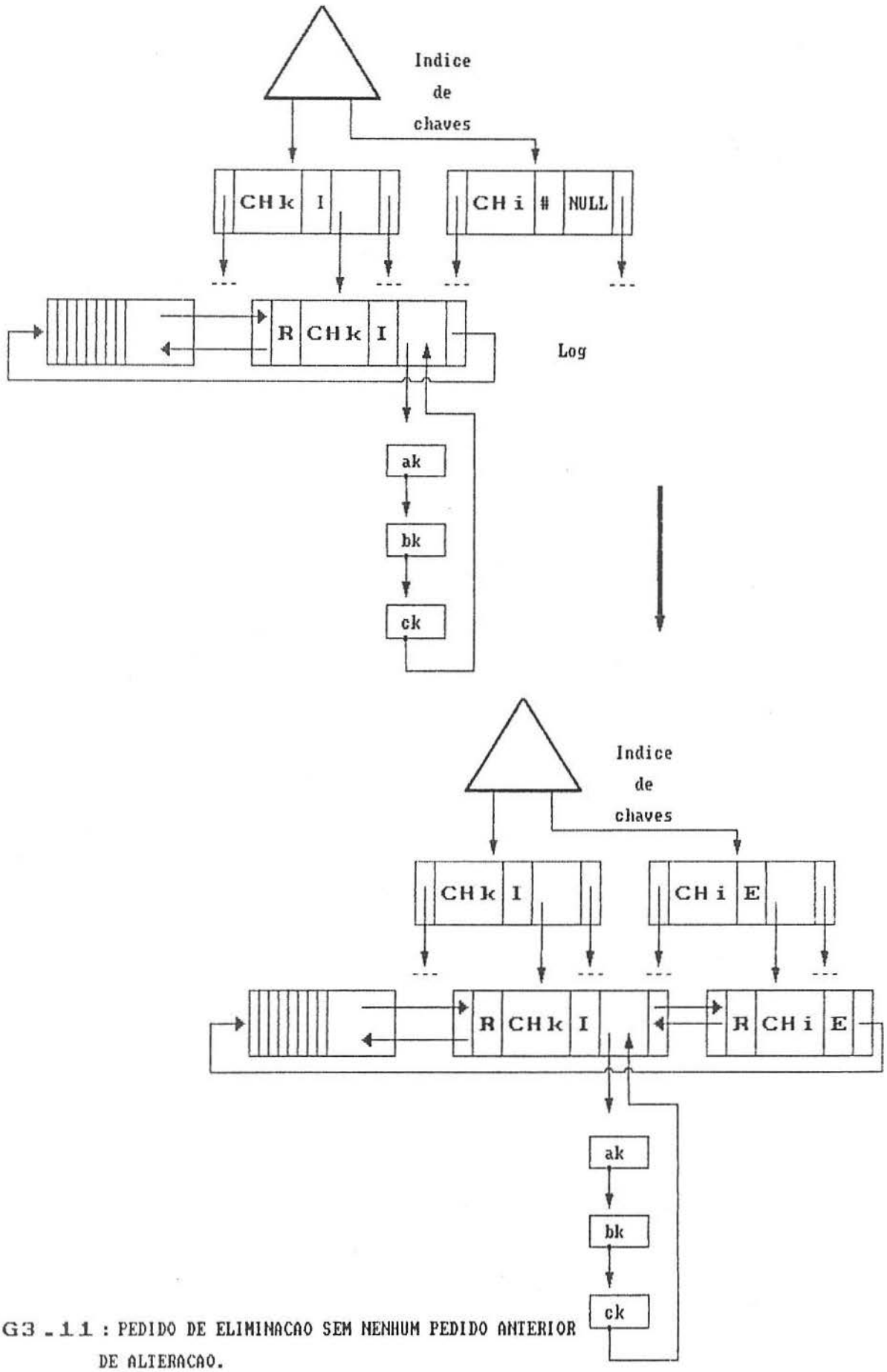

FIG3.11 : PEDIDO DE ELIMINACAO SEM NENHUM PEDIDO ANTERIOR DE ALIERACAO.

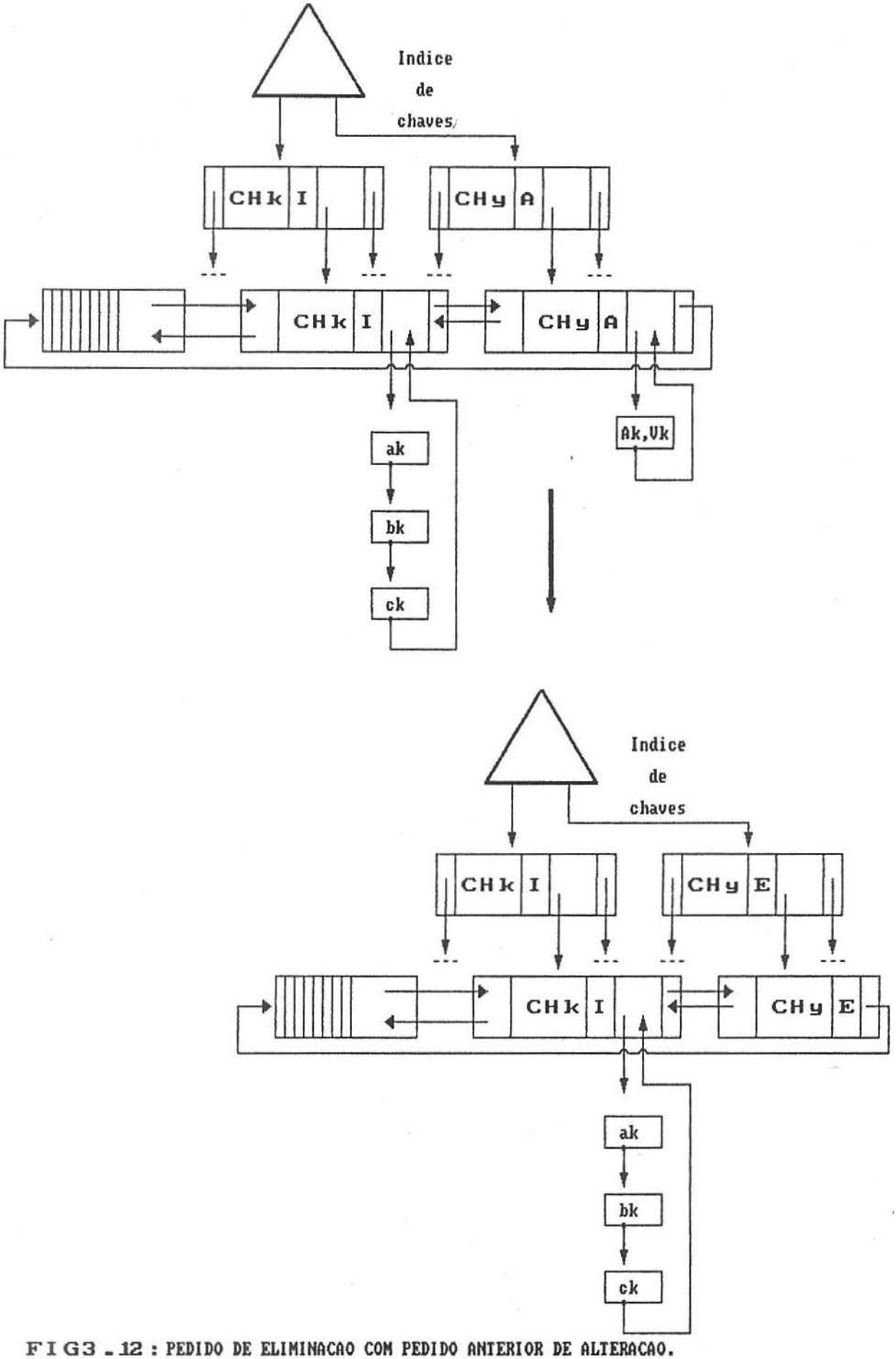

FIG3.12 : PEDIDO DE ELIMINACAO COM PEDIDO ANTERIOR DE ALTERACAO.

1A. OPERAÇÃO / 2A. OPERAÇÃO	NENHUMA	INSERÇÃO	MODIFICAÇÃO	ELIMINAÇÃO
INSERÇÃO	INSERÇÃO	-	-	INSERÇÃO
MODIFICAÇÃO	MODIFICAÇÃO	INSERÇÃO	MODIFICAÇÃO	-
ELIMINAÇÃO	ELIMINAÇÃO	REMOÇÃO	ELIMINAÇÃO	-

FIG. 3.13: REGRA PARA CONDENSAR OPERAÇÕES DURANTE O PRÉ-PROCESSAMENTO.

4. MANUTENÇÃO DE INSTANTÂNEOS E VISÕES MATERIALIZADAS.

O sistema proposto no capítulo anterior pode ser utilizado para manutenção de arquivos diferenciais para "refresh" de instantâneos. Neste caso, cada par (árvore, log) se refere a operações sobre um determinado instantâneo e não sobre uma relação. Basta adicionar uma camada de software para filtrar solicitações de atualização de relações: se uma relação a ser atualizada é utilizada na geração de um instantâneo e a tupla envolvida na operação contribui para a formação do instantâneo, a operação é passada ao sistema de pré-processamento já descrito (FIG. 4.1).

É possível assim manter cópias de instantâneos e instantâneos independentes aos quais serão aplicadas operações de "refresh", correspondentes às transações geradas pelo terceiro módulo do sistema. Este capítulo descreve a camada adicional de software que verifica se uma atualização afeta ou não um instantâneo. O sistema proposto neste capítulo será chamado de "pré-processamento para refresh" para diferenciá-lo do descrito no capítulo 3.

Enquanto todos os casos analisados na literatura se limitam a "refresh" de instantâneos gerados a partir de seleção sobre uma única relação, este capítulo mostra como utilizar as estruturas para uma classe mais abrangente de operação de geração de instantâneos.

O capítulo utiliza alguns dos resultados de Blakeley [BLA 1987] em sua tese de doutorado e que são mencionados a seguir. O sistema também pode ser utilizado na manutenção de visões materializadas, onde as operações de "refresh" não podem ser esporádicas. Neste último caso, vale a opção de atualização sob demanda: ao ser requisitada uma porção de uma visão

geradoras compostas por seleções, projeções e junção natural.

Vale observar que, dependendo da função geradora, pode haver mais de uma tupla em uma relação base que contribua para uma mesma tupla de um instantâneo, dado um determinado estado do banco de dados. Por exemplo, seja R = (ABC) com chave A e o instantâneo gerado por projeção sobre BC . Para um estado r= {(a1,b1,c1) (a2,b1,c1) (a3,b2,c2)} existe o estado do instantâneo i = {(b1 c1),(b2 c2)}. Ou seja há duas tuplas de r que contribuem para uma tupla de i. Esta observação é importante já que mostra uma das dificuldades de "refresh" de instantâneo gerado por projeção : como a chave de R não é mantida pela função geradora, uma atualização de r não é necessáriamente refletida em i. No exemplo, a eliminação de (a1 b1 c1) não afeta o instantâneo (embora esta tupla contribua para a sua formação). Além disto, a modificação de (a1 b1 c1) para (a1 b2 c1) é refletida como inserção de (b2 c1) no instantâneo.

Para que as estruturas do capítulo 3 possam ser utilizadas, é preciso que sejam feitas as seguintes hipóteses básicas sobre a formação de instantâneos:

a) nenhum instantâneo tem restrições que não sejam dependências funcionais. Ademais, nenhum instantâneo tem restrições adicionais às "herdadas" das relações que o geram, exceto nos casos c2) e c3).

b) a integridade de um instantâneo depende exclusivamente da integridade de suas relações base. Desta forma, qualquer operação aceita para uma relação base pode ser propagada para o instantâneo sem necessidade de verificação adicional de integridade que não seja a imposta por sua função geradora.

c) todo o instantâneo tem um identificador de suas tuplas, para isso considerado sua (única) chave:

c1) se o instantâneo tem uma única relação base, e sua

função geradora inclui uma das chaves desta relação: está chave é também o identificador do instantâneo.

c2) se o instantâneo tem uma única relação base e nenhuma chave é incluida (função de projeção): cabe ao administrador do banco de dados definir o conjunto de atributos que servirão de chave ao instantâneo. No pior caso, cada tupla é o próprio identificador.

c3) o instantâneo tem mais de uma relação base (gerado por junções de relações). Só serão considerados instantâneos com mais de uma relação base se não houver projeção sobre o resultado e a junção for sem perdas. Neste caso, a chave do instantâneo é a chave da relação resultante.

Note que C3 se refere a uma junção sem perdas onde a relação resultante pode não estár nem mesmo em 3NF, o que contraria uma das hipóteses do sistema de pré-processamento. No entanto, como se supõe (hipótese b) que as relações base são consistentes, não é preciso processamento adicional para verificar a consistência do instantâneo (ou seja, verificação de dependências transitivas).

EXEMPLO: Sejam R1 = (ABC), chave A e R2 = (CD), chave C relações base do instantâneo I = R1 R2. Então a chave de I é A e ele tem a dependência (transitiva) A -> C -> D. A princípio, inserções ou modificações no instantâneo deveriam verificar as dependências transitivas. Como, no entanto, essas operações são fruto de atualizações nas relações base, isto não é necessário.

As hipóteses feitas são semelhantes às utilizadas no tratamento dado à manutenção de visões quando se atualiza suas relações base [KEL 1986].

A tese de [BLA 1987] define formalmente quando atualizações sobre relações base afetam visões materializadas sobre estas

relações. Alguns dos resultados de [BLA 1987] que serão usados nesta tese são descritos a seguir. Alguns de seus teoremas são simplificados, tendo em vista que o universo de funções geradoras consideradas aqui é mais reduzido que o de Blakeley.

O autor busca manter as visões materializadas consistentes de forma a evitar que sejam regeradas a cada atualização das suas relações base. Desta forma, visões materializadas funcionam como uma espécie de instantâneo, sempre mantido consistente (não há "refresh" periódico). Blakeley determina as seguintes situações:

a) condições necessárias e suficientes para a caracterização de atualizações irrelevantes.

Uma atualização de uma relação base é irrelevante quando não afeta a visão materializada (ou, no nosso caso, o instantâneo). Estas condições independem do estado do banco de dados.

b) caracterização de atualizações computadas autonomamente. Estas são atualizações a relações base cujo reflexo na visão materializada pode ser determinado usando apenas a operação de atualização, a definição da visão e o conteúdo da visão (sem se preocupar com o estado da relação base). O autor diferencia entre atualizações computadas autonomamente de forma incondicional (quando o estado do banco de dados não interessa nunca) ou condicional (quando o estado do banco de dados pode interessar). No último caso, define em que situações se pode ignorar o estado do banco de dados. Define condições necessárias e suficientes para uma atualização ser do tipo incondicional e para inserções e eliminações serem do tipo condicional.

c) tratamento de atualização diferencial de visões materializadas utilizando pré-processamento de conjuntos de operações (de forma semelhante à otimização de consultas múltiplas descrita no capítulo 1).

Quando é considerado que definições de instantâneos não

envolvem junções, as modificações de um instantâneo são um subconjunto de modificações feitas na tabela base correspondente e podem ser tratadas tupla a tupla.

Esta tese utilizará alguns dos resultados das partes (a) e (b). Assim como [BLA 1987], será suposto que as únicas funções geradoras de instantâneos aceitáveis são expressões do tipo PSJ, ou seja, formadas por combinações de projeções, seleções e junções. Como comentado por Blakeley, um resultado da teoria relacional é que expressões PSJ podem ser tranformadas em expressões da forma produto cartesiano seguido de uma seleção seguido de uma projeção. A condição de seleção corresponde ao conjunto de seleções e condições de junção da expressão PSJ.

As expressões PSJ consideradas nesta tese para geração de instantâneos são:

a) para uma única relação base: uma seleção seguida opcionalmente por uma projeção

b) para mais de uma relação base: produto cartesiano das relações base seguido de uma seleção, ou, equivalentemente, junção natural das relações base seguido opcionalmente por uma seleção.

Outras hipóteses feitas para esta tese que permitam simplificar e adotar apenas alguns dos resultados de [BLA 1987] são:

* supõe-se que os pedidos de atualização sejam analisados sobre uma tupla de uma relação de cada vez (ou seja, ignoram-se transações e atualização multi-relações)

* as relações base estão sempre disponíveis para que o SGBD possa verificar a validade de uma atualização. Apenas atualizações válidas são posteriormente processadas pelo sistema aqui proposto.

* a função geradora não admite junção de uma relação consigo mesma.

* a inicialização do sistema de pré-processamento para "refresh" pressupõe que todos os instantâneos e sua cópias reflitam o estado mais recente de suas relações base. Desta forma, é indiferente se o par (árvore,log) gerado para um instantâneo I é criado a partir de alguma cópia de I ou a partir da materialização de I no nó de processamento dadas sua relações base. Para aproveitar o sistema do capítulo 3, é óbvio que a geração destes pares será feito a partir dos instantâneos.

* será gerado apenas um par (árvore, log) para cada função geradora de instantâneo. Todas as cópias do instantâneo correspondente deverão ser atualizadas a partir do mesmo par.

* consultas a instantâneos não causarão "refresh". O "refresh" será aplicado mediante solicitação ou quando o log do instantâneo estiver cheio.

* Consultas a uma visão materializada deverão ser processadas da seguinte forma: o resultado da consulta é avaliado e verifica-se se há atualização pendente. Se houver, as tuplas já atualizads são mostradas ao usuário e a seguir é efetuada a atualização na visão.

4.2 PRÉ-PROCESSAMENTO PARA "REFRESH".

Seja R um esquema, t uma tupla atualizada (inserida, eliminada ou modificada) em r. Sejam Il...Ij os instantâneos a serem mantidos com funções Gl...Gj. A notação at(Ik) denota um conjunto (possívelmente vazio) de atualizações no instantâneo Ik causadas pela atualização de t, denotada AT(t), onde AT = INS, EL

ou MOD.

O algoritmo de pré- processamento para "refresh" é o seguinte:

Passo1: Determinar o conjunto de instantâneos T = {IA...Ik} para os quais R é relação base.

Passo2: Para cada Ij em T, determinar at(Ij)

Passo3: Aplicar as operações de at(Ij) ao par (árvore, log) do instantâneo Ij, utilizando o sistema do capítulo 3.

Valem as seguinte observações:

* algumas das operações at(Ij) podem ser supérfluas (por exemplo, inserção de tupla já existente no instantâneo). A seção a seguir tenta definir o passo 2 de forma a evitar tais operações.

* caso Gj contenha uma projeção que não mantenha a chave da relação base, é preciso alterar a estrutura da árvore de Ij. Este caso é analisado à parte. A alteração consiste na colocação no índice de um contador indicando quantas tuplas da relação base dão origem à tupla correspondente no instantâneo.

* caso at(Ij) contenha mais de uma atualização sua ordem de execução no passo 3 não importa para o resultado final. Como se verá a seguir, at(Ij) corresponde a um conjunto de atualizações somente quando Gj contém uma junção. Neste caso, como a operação junção é monotônica, projeções não são aceitas e o instantâneo não tem restrições adicionais, vale o seguinte:

- uma inserção em r corresponde a zero ou mais inserções em Ij
- uma eliminação em r corresponde a zero ou mais eliminações em Ij

- uma modificação em r (lembrando que só se admite modificação de elementos não chave) corresponde a zero ou mais modificações em tuplas distintas de Ij.

A próxima seção descreve a determinação de at(Ij).

4.3 DETERMINAÇÃO DAS ATUALIZAÇÕES DIFERENCIAIS at(Ij)

Caso a) Gj é a identidade (ou seja, Ij = R).

Então o conjunto de atualizações no instantâneo Ij, at(Ij), causadas pela atualização de t, será igual ao conjunto de todas as atualizações de t da relação base.

Caso b) Gj é uma a operação PS (uma projeção e uma seleção)

b1) Gj = $\sigma_c R$, onde c é uma condição.

Então o conjunto de atualizações será igual ao conjunto de atualizações para t tal que t satisfaz a condição C.

b2) Gj = r[x] e $k \subseteq x$ para k chave de R.

Então o conjunto de atualizações no instantâneo Ij, at(Ij) será igual ao conjunto das atualizações para t que afetam os atributos X .

b3) Gj = $\sigma_c r[x]$ onde c e x são definidos em b1) e b2).

O conjunto de atualizações no instantâneo Ij, at(Ij), será igual ao conjunto de atualizações para t tal que os atributos
X de t satisfaçam a condição C .

b4) Gj = r[x] e não existe chave K de R tal que K X.

O instantâneo tem um identificador ID $\subseteq$ X, sendo utilizado na árvore como contador de tuplas.

Seja n o número de tuplas t' que contribuem para geração

de tj pertencente a Ij e X é atributo de t'.

As operações sobre a tupla tj do instantâneo com identificador ID são

EL_COND (tj):

Se n > 1 então n <- n - 1

else Se n = 1 então at(Ij) = EL(tj),

n <- 0,

else erro.

INS-COND (tj)

Se n > 0 então n <- n + 1 senão n <- 1, at(Ij) = INS(tj)

MOD(tj) = modificação de valor não pertencente a ID: at(Ij) = MOD(tj)

Seja o pedido AT(t):

Processá-lo da seguinte forma:

INS(t) = processar INS_COND(tj)

EL(t) = processar EL_COND (tj)

MOD (t) = se valor modificado não pertence a ID, processar como MOD(tj), se o valor modificado pertencer a ID, procesar como (EL_COND (tj), INS_COND (tj))

b5) Gj = $\sigma_C r[x]$ onde C e X são definidos em b1) e b4).

Então o conjunto de atualizações no instantâneo Ij, at(Ij), será igual ao conjunto gerado por b4) menos as tuplas que não satisfazem a condição C.

Caso c) Gj igual a operação SJ (uma seleção e junções naturais).

Então o conjunto de atualizações no instantâneo Ij, at(Ij), será igual ao conjunto gerado por b1) considerando o conjunto de tup gerado pela função junção.

Formalmente temos: seja uma atualização AT(t), onde t r afeta Ij. Se o instantâneo é cópia da relação, at (1j)= AT(t).

a) casos onde função geradora G é do tipo PS (projeção e seleção sobre r)

a1) $G = \sigma_c R$ (c é um predicado sobre os atributos de R)

Se $t \in \sigma_c r$, então $at(Ij) = AT(t)$; senão, AT(t) é irrelevante [BLA 1987].

a2) $G = r[x]$ e $(\exists K / K \rightarrow R \wedge K \subseteq X)$ (alguma chave de R é mantida)

Se AT(t) é alteração de atributos Y e $Y \cap X = \phi$, então AT(t) é irrelevante; senão, at(Ij) = AT(t[x]).

a3) $G = r[x]$ e $(\neg\exists K / K \rightarrow R \wedge K \subseteq X)$ (nenhuma chave de R é mantida).

Neste caso o instantâneo tem um identificador $ID \subseteq X$ e é utilizada a árvore com contador de tuplas nos nós (FIGS. 4.2 e 4.3 e 4.4).

Se AT(t) = eliminar/inserir(t) então at (Ij) = EL-COND/INS-COND(t[x]), onde EL-COND e INS-COND são procedimentos de remoção ou inserção na árvore condicionados ao valor do contador associado a (t[x]): a tupla deve ser removida se o valor do contador for 1 e inserida se não existir. Caso contrário, basta atualizar o contador.

Se AT(t) é alteração de atributos Y e $Y \cap X = \phi$, então AT(t) é irrelevante, senão at(Ij) = {EL-COND(t[x]), INS-COND(t'[x])} sendo t' a tupla modificada.

a4) $G = \sigma cr[x]$

Se $t \in \sigma cr$, aplicar regras a2 ou a3; senão, AT(t) é irrelevante.

b) Casos onde a função geradora é SJ (junção seguida de seleção)

b1) $G = R1 \bowtie \ldots \bowtie R2 \bowtie \ldots \bowtie Rn$ e $(\exists i / R = Ri)$

Neste caso, at(t) não é computável autonomamente [BLA 1987]

rn),

sendo at(Ij) = AT(t).

b2) G = σc(R1$\bowtie$...$\bowtie$R2$\bowtie$...$\bowtie$Rn) e ($\exists$i/ R = Ri)

Se a condição envolver atributos de Ri, gerar ρ =(r1$\bowtie$...t$\bowtie$...$\bowtie$rn) e ($\forall$ v/v$\in\rho\wedge$v$\in\sigma$cρ) aplicar at(Ij) = AT(v). Senão AT(v) é irrelevante.

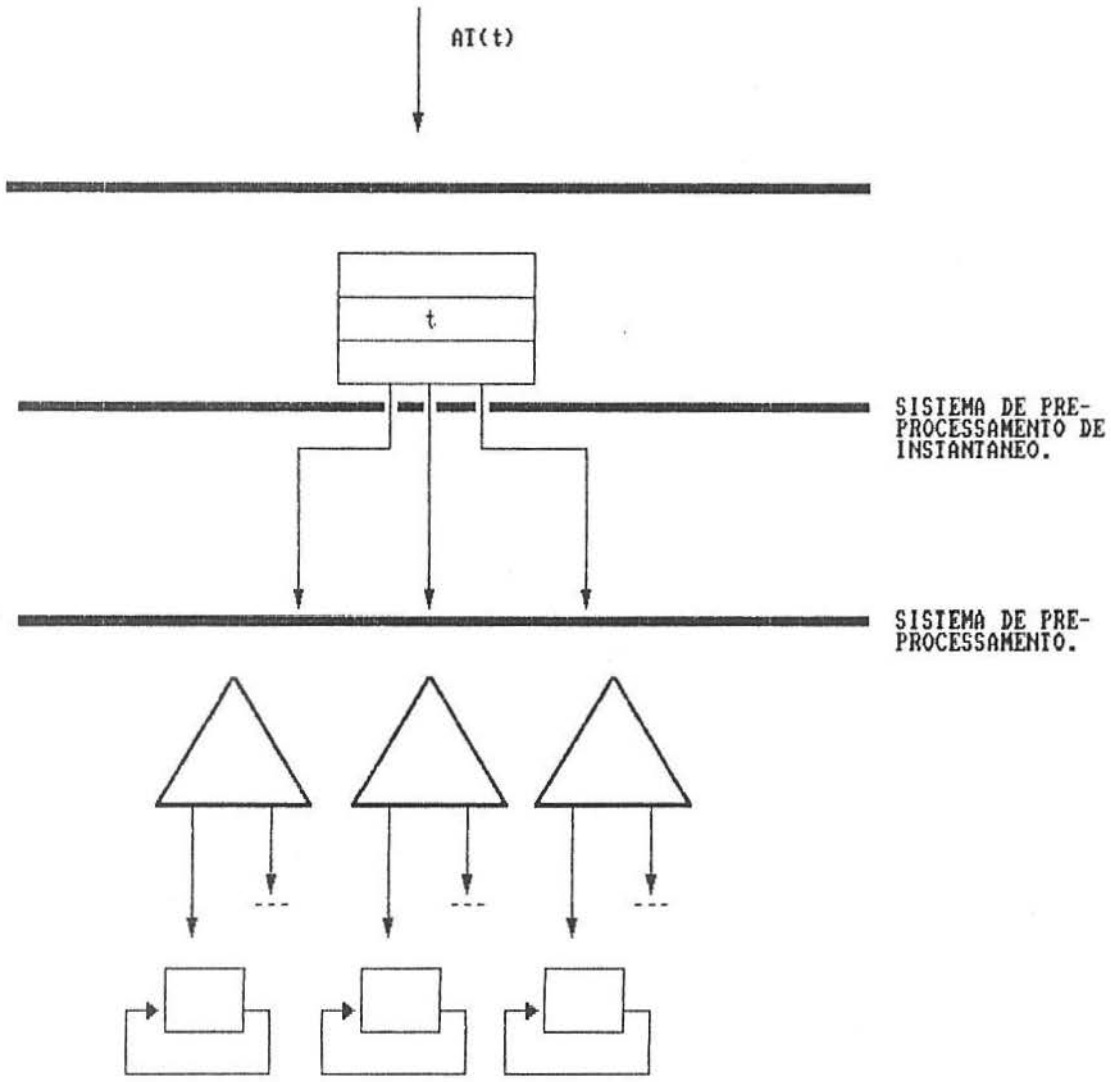

FIG4_1 : CAMADA DE SOFTWARE.

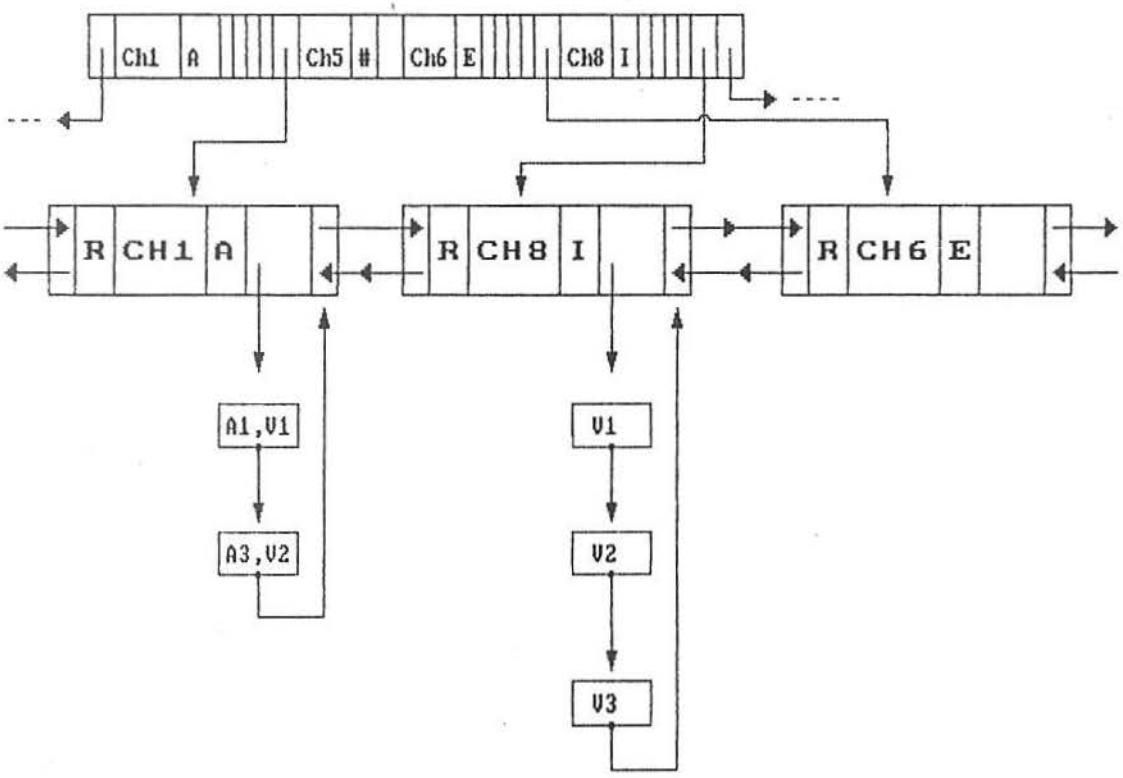

FIG4.2: ENTRADAS DO LOG: TUPLA COM CHAVE CH1 TEM ATRIBUTOS A1 E A3 ALTERADOS PARA VALORES V1 E V2; TUPLA DE CHAVE Ch8 INSERIDA, COM VALORES NAO CHAVE V1,V2,V3. TUPLA DE CHAVE Ch6 ELIMINADA. A PARTE HACHURADA DE UM NO DA ARVORE CORRESPONDE AO CONTADOR.

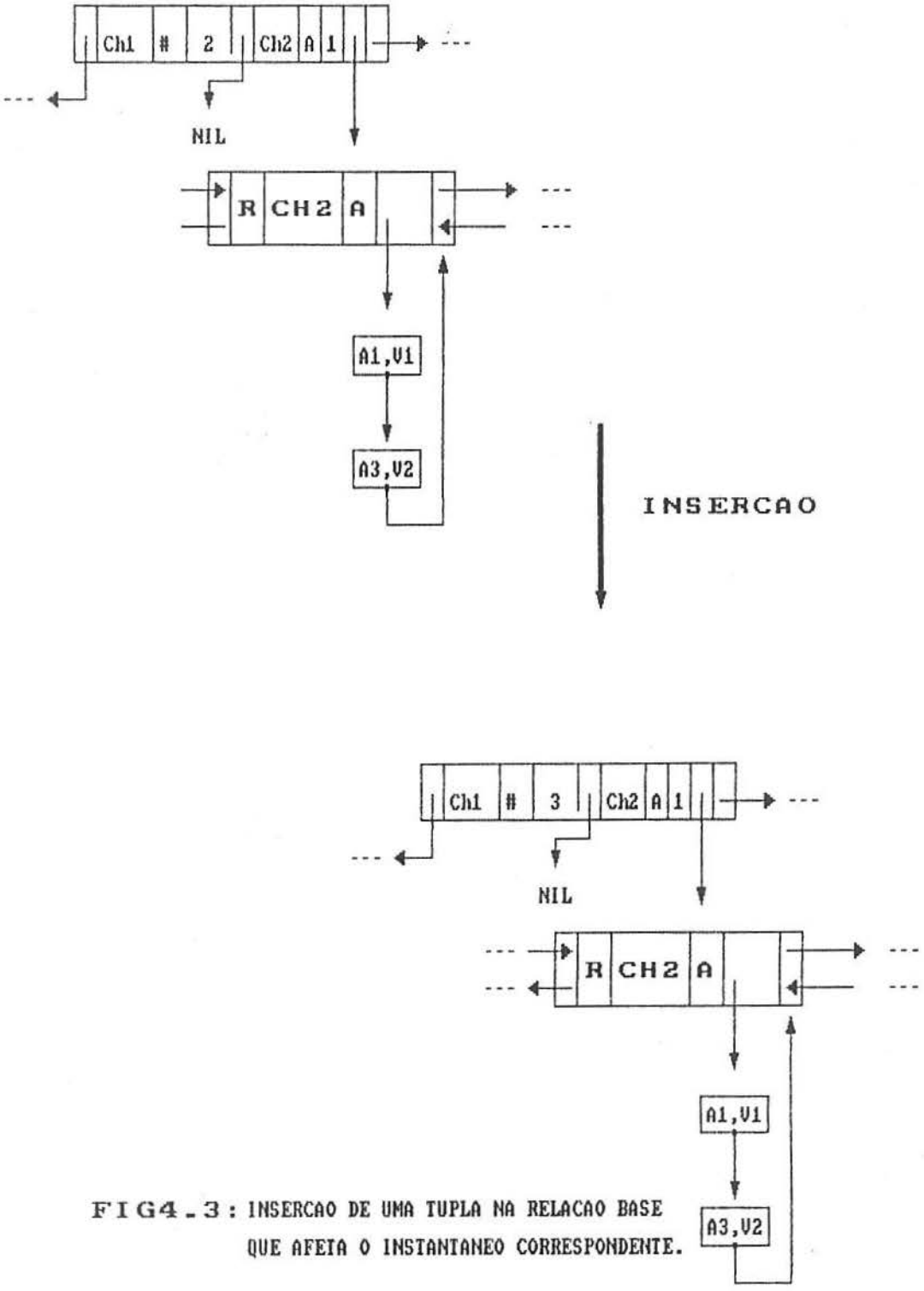

FIG4_3: INSERCAO DE UMA TUPLA NA RELACAO BASE QUE AFETA O INSTANTANEO CORRESPONDENTE.

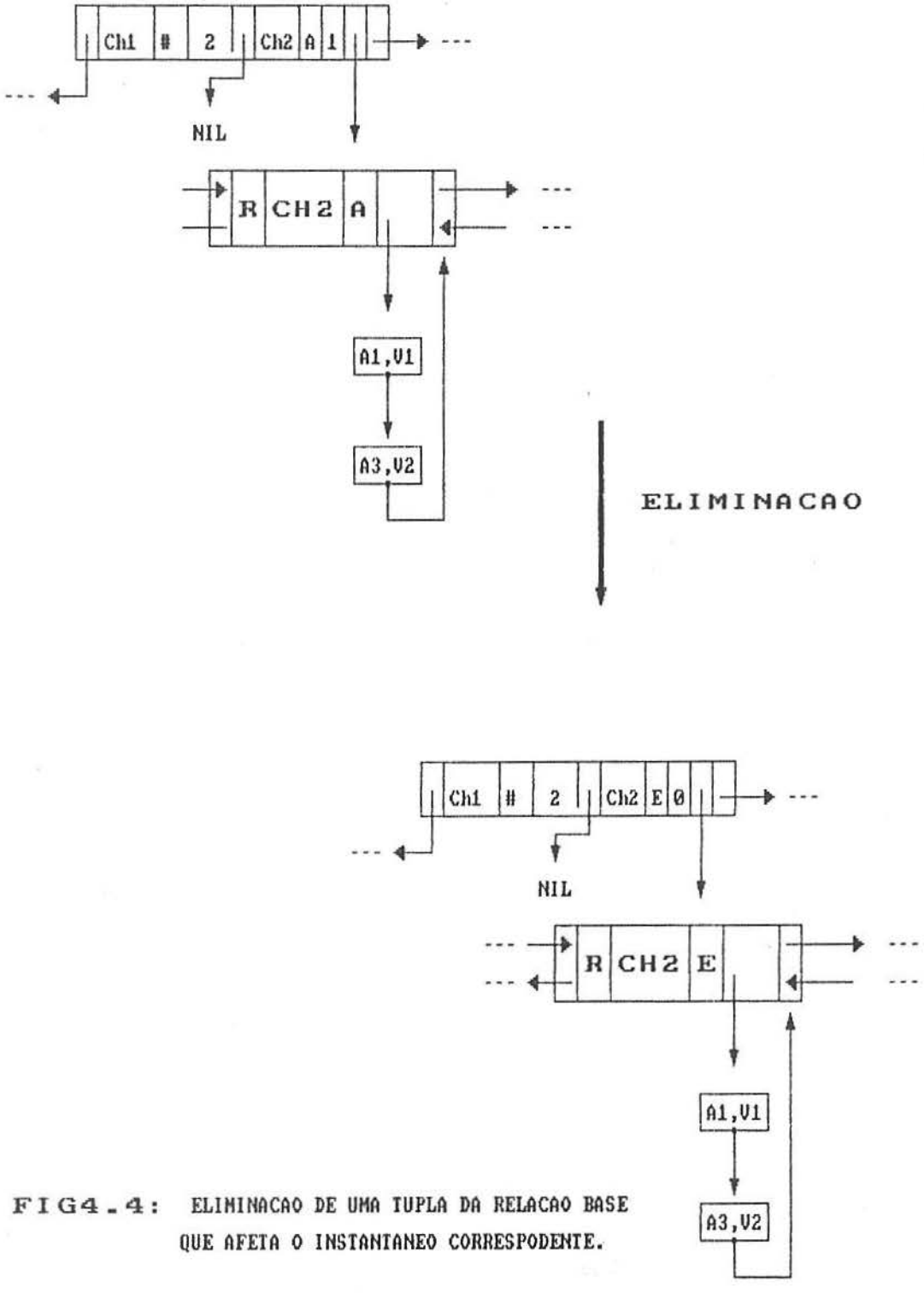

FIG4.4: ELIMINACAO DE UMA TUPLA DA RELACAO BASE QUE AFETA O INSTANTANEO CORRESPODENTE.

5.TESTES E ESTRUTURAS ALTERNATIVAS.

5.1 PROTÓTIPO IMPLEMENTADO.

O sistema foi implementado em PASCAL 3.4com o SGBD RDB em um VAX 11/785, havendo sido feitas as seguintes simplificações:

1) Apenas uma relação é atualizada;

2) A interface para a chegada de solicitações de atualização é diretamente feita entre o usuário e o sistema de pré-processamento sem a interferência do SGBD. O SGBD só é acionado no momento de processar as transações (MÓDULO 3) e manter as estruturas (MÓDULO 1).

O sistema recebe os pedidos de atualização através de uma entrada que possui o seguinte formato:

Em primeiro lugar o usuário informa ao sistema o tipo de operação desejada:

- Inserção ("STORE");
- Modificação ("MODIFY");
- Eliminação ("ERASE").

Em seguida o sistema pede o nome da relação, o nome e o valor do atributo chave (aqui o sistema verifica se o pedido é válido). O sistema ainda deve saber:

1) Se o pedido for de inserção de uma nova tupla:
   a) número de campos que a tupla possui;
   b) nome dos atributos seguido dos seus valores;
2) Se o pedido for de modificação:
   a) Nome do atributo seguido do seu valor.

No caso de inserção de tupla, usou-se a mesma rotina da modificação, ou seja, o log contém um registro que aponta para

uma lista formada por pares (nome do atributo, valor do atributo). No caso geral, isto causa grande desperdício de espaço já que a inserção pelo especificado no capítulo 3 obedece ao esquema fixo da relação.

O sistema foi testado utilizando uma relação real do CPqD. Os testes envolveram misturas de vários tipos de atualização (arquivo contendo cem pedidos de atualização), sendo geradas 32 transações de atualização. As misturas visaram testar a condensação. Além disto, foram feitos testes com um único tipo de atualização (só modificações ou só inserções) para testar a carga de criação e manutenção de log. Os resultados de desempenho não foram conclusivos, já que seria necessário comparar este sistema com processamento não condensado de atualização, bem como utilizá-lo em ambiente distribuido. Em outras palavras, um estudo comparativo bem feito exigiria utilização de ferramentas específicas (por exemplo monitores).

5.2 COMPARAÇÃO ENTRE AS ESTRUTURAS DE DADOS UTILIZADAS E ALGUMAS ALTERNATIVAS.

5.2.1 ÍNDICE DE CHAVES

Esta tese optou pela implementação do índice de chaves segundo uma árvore-B. Uma outra alternativa seria a de manter o índice em uma tabela de "hash", o que garantiria um acesso mais rápido às chaves da relação. Entretanto, a escolha de funções de

"hash" utilizadas poderia trazer problemas, sob o ponto de vista de que a função a ser escolhida deveria variar a cada relação para minimizar o problema de colisões, considerando-se um programa geral para processar qualquer conjunto de relações.

Além disso, deveria ser permitido o "rehash" nos casos da relação associada ao índice de chaves ter seu tamanho muito aumentado ou diminuído. Idealmente, no caso de haver várias relações deveria existir diferentes tipos de funções (uma para cada relação) de modo a auferir as vantagens da utilização desse método.

Assim, para evitar esse tipo de problema, optou-se pela utilização de árvores-B para manutenção do índice de chaves.

Decidiu-se manter uma árvore completa por relação, isto é, contendo todas as chaves da relação para permitir a detecção precoce de algumas violações de integridade. Esta opção de implementação permite que se recuse algumas atualizações inválidas mesmo que venha a ocorrer uma queda na rede (por exemplo, inserção de tuplas já existentes). Uma alternativa mais eficiente em termos de espaço seria manter na árvore apenas chaves referentes a atualizações solicitadas e não processadas, cujas chaves seriam inseridas na árvore à medida em que fossem chegando os pedidos. Esta segunda alternativa dificultaria o reconhecimento de atualizações invalidas (como por exemplo modificação de tuplas não existentes). A árvore completa é uma opção racional quando se trata de processar instantâneos (capítulo 4), que não sejam cópias de relação. Neste caso, o conjunto de chaves é um subconjunto das chaves da(s) relação(s) base. Caso haja poucas atualizações entre dois "refresh", a estrutura completa é ainda pouco econômica em termos de espaço. Por outro lado, economiza entradas no "log", que caso contrário

precisa ter informações sobre estado anterior de cada tupla acessada [KAH 1987]

Uma outra alternativa para o índice seria ter uma árvore onde apenas as folhas seriam páginas de apontadores para o log (árvore-B+). A vantagem dessa organização é que ela oferece maior eficiência no aproveitamento de espaço e na execução de atualizações na árvore, permitindo pesquisa sequencial ordenada de chaves. Em contrapartida existe o problema do manuseio de páginas: para acessar a página, a árvore deve ser percorrida, sendo a busca da chave na página feita sequencialmente. Uma consulta se dá a partir da raiz através do índice até as folhas. Uma vez que todas as chaves residem nas folhas, não se deve levar em conta os valores encontrados ao longo do caminho até a folha corrente para responder à consulta. Além disto, as páginas podem não ter o mesmo tamanho.

A opção adotada por esta tese foi a de manter a estrutura da árvore B, com nós contendo as chaves da relação, os códigos de atualização e os apontadores para o log de atualizações. Embora esta estrutura seja menos eficiente em termos de espaço, ela atende melhor aos requisitos de manutenção e acesso ao log de atualizações, onde os registros não estão ordenados e nem têm tamanho fixo.

5.2.2 LOG DE ATUALIZAÇÕES

O log foi armazenado em uma lista duplamente encadeada.

Uma das alternativas quanto à manutenção dos registros do log seria a de utilizar alocação sequencial (log tabular de

tamanho fixo). No entanto, isso seria ineficiente, pois cada vez que ocorresse uma inserção ou eliminação os registros teriam que ser rearranjados. Uma outra solução seria mantê-lo em uma lista circular, que resolveria o problema de reorganização, mas no caso de haver uma eliminação de um registro aleatório seria necessário o conhecimento de pelo menos dois apontadores, o do próprio registro a ser eliminado e o do elemento anterior para poder fazer a concatenação com elemento seguinte ao eliminado. Embora haja soluções mais econômicas em termos de ponteiros (por exemplo, cadeia coral, um tipo de cadeia mista em que alguns registros são duplamente encadeados), adotou-se uma estrutura sob a forma de uma lista circular duplamente ligada, para solucionar o problema anterior. Além de possibilitar a navegação pelo **log** em ambos os sentidos, facilita a atualização de seus elementos e os algoritmos para a sua manutenção são bem mais simples.

O **log** de atualizações, além dos registros que contêm as chaves das relações, possui apontadores que indicam as atualizações propriamente ditas. Em particular, para as operações de alteração e inserção, foi adotada uma lista ligada, isto é, para as operações de inserção e alteração do registro do log possui um apontador que fornece o endereço da lista que contém a cadeia de atributos, (nome e valor do atributo). O tamanho máximo da lista, para uma relação, é o número de atributos da relação. Essa solução foi adotada para acomodar o caso de existir mais de uma relação, com tuplas de tamanhos diferentes. O protótipo implementado utiliza um tipo definido pelo PASCAL , **varying of char**, que permite cadeias de elementos de tamanho variável, embora a priori seja alocado espaço para o tamanho máximo que a variável pode assumir.

Essa estratégia foi adotada visando linguagens de

manipulação de dados (como o RDB) que exigem que para as operações de inserção e modificação haja especificação do nome de cada campo. No caso de se utilizar uma linguagem que aceite inserção sem definição de campos (isto é, INSIRA "string") é possível supor um tamanho máximo de registro, sendo este passado para o log. Para operações de alteração se o pedido de atualização envolver apenas um campo da tupla, essa lista será constituida de um único elemento; se envolver dois campos será constituida de dois elementos e assim por diante. Essa solução economiza espaço: se as atualizações fossem mantidas em registros fixos, existiriam campos vazios (já que nem todas as atualizações envolvem todos os atributos de tuplas). Por exemplo, se uma determinada relação possui uma tupla de cinco atributos (ordem máxima), e ocorre um pedido de atualização que envolve apenas alteração de um atributo, a opção adotada na tese será de uma lista formada por apenas um elemento enquanto que o registro fixo possuiria quatro campos desperdiçados.

5.2.3 LOG ÚNICO.

A tese propõe uma árvore e um log por relação. Uma opção seria uma árvore por relação e um log único de atualização onde cada elemento seria acrescido do nome da relação a ser atualizada. Esta última opção pode apresentar vantagens no processamento do módulo 3, já que bastaria gerar transações a partir de uma única lista. No entanto, é possível que nem todas as relações devam ser atualizadas ao mesmo tempo, o que exigiria processamento seletivo do log. Ademais, esta opção traria

problemas no processamento das árvores índice (vide 5.2.1).

Cabe ainda mencionar que a opção do log único aumentaria o espaço gasto pelo log. A opção de um log por tabela é a mesma de [KAH 1987].

5.3 PROCESSAMENTOS ALTERNATIVOS.

5.3.1 LIBERAÇÃO DO LOG.

São permitidos dois tipos de liberação total do **log**:

a)Fim de processamento: a árvore é processada, o log é liberado sequencialmente e em seguida a árvore também é liberada.

b) log de atualizações cheio: Neste caso, foram estudadas alternativas para liberação do **log**, sendo desejável manter a árvore. Estas alternativas são indicadas a seguir

b1) Modificar o registro do **log** para que contenha um campo que aponta de volta para o nó correspondente na árvore-B. Neste caso, para liberar o **log** basta percorrer os seus registros e atualizar os respectivos nós da árvore-B com o código adequado (ou seja, indicando que não existem pedidos de atualizações anteriores).

b2) A cada registro do **log** a ser liberado, procura-se o correspondente na árvore e atualiza-se o código de operação. Através do campo chave do registro do **log**, procura-se o campo com a chave correspondente na árvore-B e o campo código é atualizado indicando que não existem atualizações pendentes. Esta opção é

muito demorada, pois exige que a árvore seja percorrida várias vezes.

b3) Liberar **log** através do processamento da árvore, ou seja, como a árvore possui apontadores para o **log**, à medida que ela é processada os registros do **log** podem ser liberados e os respectivos campos códigos atualizados.

A opção b3 é melhor no sentido de que a quantidade de ponteiros é reduzida, mas possui a desvantagem de precisar percorrer a árvore inteira para atualizar os códigos. Por outro lado se a árvore for do tipo reduzido (discutido em 5.2.1, só contendo informações sobre tuplas a serem atualizadas), é possivel eliminar o **log** sequencialmente sem haver a necessidade de se preocupar com a árvore.

5.3.2 ESTRUTURA MULTI-RELAÇÃO.

A estrutura proposta para permitir processamento de multi-relações apresenta o inconveniente de armazenar o esquema de cada relação, o que sempre pode ser obtido diretamente do SGBD. No entanto, este desperdicio (pequeno) de espaço é compensado pela economia em evitar uma consulta via DML ao esquema a cada acesso ao log.

6. CONCLUSÕES E EXTENsÕES.

Esta tese discutiu o problema de condensação e diferimento de atualizações em bancos de dados relacionais, principalmente no que diz respeito a bancos de dados distribuidos, com a possibilidade de várias cópias de uma mesma relação distribuídas ao longo da rede. A tese apresentou um sistema para efetuar a condensação de atualizações. O sistema recebe solicitações de atualizações, efetua seu pré-processamento e gera, periódicamente transações já sob a forma executável pelo SGBD. O sistema foi desenvolvido de forma modular, de modo a torná-lo independente na medida do possível de um SGBD específico.

Vale salientar que esta independência nem sempre é benéfica. No caso das estruturas de dados, por exemplo, redundou na criação de índices adicionais quando os próprios índices do sistema poderiam ter sido utilizados com consequente economia de espaço. Optou-se, assim, pela independência em troca de maior tempo de processamento inicial (no preparo dos índices) e de aumento de espaço necesário para o sistema funcionar.

Dentre as vantagens de um sistema deste tipo podem ser citadas a economia de processamento na condensação (supondo que não haja consultas entre duas atualizações de uma mesma tupla), bem como o suporte à manutenção de cópias de uma mesma relação. No primeiro caso, pode-se imaginar que uma atualização só seja aplicada no momento em que uma tupla é consultada. A consulta é precedida por acesso às estruturas de árvore e log; caso haja atualização (modificação) pendente, a transação correspondente é gerada e passada ao SGBD, que a efetua e responde à consulta a

seguir. No segundo caso, se um dos nós onde reside uma cópia não estiver conectado à rede, o SGBD pode postergar as transações geradas pelo sistema aqui proposto, para executá-las quando houver recuperação, enquanto aplica as transações às demais cópias. Desta forma, já que todas as cópias são atualizadas a partir do mesmo conjunto de transações, garante-se a consistência do sistema.

A tese mostrou, no capítulo 4, como o sistema pode ser utilizado para "refresh" de instantâneos e manutenção de visões materializadas. Enquanto os trabalhos pesquisados na literatura se limitam a discutir estruturas de "refresh" de instantâneos gerados por seleção, a tese generaliza a discussão para casos em que os instantâneos são gerados por seleção e projeção, abordando também tratamento de alguns tipos de instantâneos gerados por expressões PSJ.

Finalmente, o capítulo 5 discute o protótipo implementado e justifica algumas das decisões de implementação, além de prover estruturas alternativas.

O sistema proposto para condensação e diferimento não é recomendável para todos os casos, já que há situações em que pode impactar o desempenho. Este sistema é recomendável principalmente nas seguintes situações: ambientes com muitas transações de atualização sobre um conjunto pequeno de dados (caso em que a manutenção das estruturas não ocuparia espaço e haveria economia de atualizações intermediárias); ambientes com transações de atualizações extensas (a manutenção do log evitaria que muitos arquivos de dados fossem trancados durante a transação, adiando esta atividade até o fim da transação); ambientes com poucas atualizações (as atualizações podem ser adiadas até ser feita consulta aos dados modificados); ambientes distribuidos onde haja

quedas em nós em arquivos replicados (caso em que o sistema funcionaria como backup). As estruturas podem também ser utilizadas para a manutenção de visões materializadas [BLA 1987].

Em alguns outros casos, é preciso um estudo de desempenho para verificar o impacto do sistema. Um exemplo é um ambiente em que há grande número de atualizações, porém distribuídas uniformemente sobre todos os dados. Nesta última situação, é possível que os custos de manutenção as estruturas não compensem as vantagens de diferimento, se não conseguir condensação.

Finalmente, vale salientar que as estruturas propostas podem ser consideradas arquivos diferenciais para facilitar a consulta. Assim, ao se consultar dados, o resultado da consulta pode ser calculado, modificado a partir das informações das estruturas, e só então apresentado ao usuário. Desta forma, não haveria necessidade de aplicar a atualizaçào nos arquivos quando efetuada uma consulta: as atualizações podem ser adiadas em função da decisão do administrador de banco de dados. Neste contexto, as estruturas funcionariam como em [TUC 1988], como uma espécie de arquivo diferencial a ser aplicado ao resultado de consultas.

Uma extensão ao sistema implementado na tese seria a implementação da camada adicional proposta no capítulo 4, que permitiria o processamento de instantâneos. Outra extensão seria a implementação de algumas estruturas alternativas discutidas no capítulo 5, visando o estudo comparativo de desempenho.

Esta tese apresenta um protótipo, que parte do princípio de que os tipos de dados definidos nas relações do banco de dados são **varying of char**. Outra aplicação poderia exigir chaves de outro tipo. Uma alternativa seria pré-processar relações para tranformar suas chaves para **char** e depois quando da atualização fazer exatamente o inverso.

Um outro tópico que merece maior atenção é o processamento de atualização sob demanda: atualizações seriam condensadas e diferidas até o momento de consulta à(s) tupla(s) envolvida(s), quando então seriam aplicadas. As estruturas propostas talvez não sejam as mais adequadas para este tipo de tratamento. Além disto, há que considerar o possível "overhead" de consulta às estruturas para verificar atualizações pendentes a cada consulta ao banco de dados. No protótipo implementado, este passo limitou-se à geração da transação correspondente, com liberação da entrada no log. No entanto, no caso de múltiplas cópias de uma mesma relação, talvez não seja conveniente realizar a atualização sobre todas as cópias. Neste caso, não faria sentido liberar a entrada no log.

O sistema pressupõe que é acionado pelo SGBD, que lhe passa as informações necessárias a cada pedido de atualização/consulta. No entanto, a implementação do protótipo recebe as solicitações diretamente do usuário e se comunica com o SGBD para geração de estruturas (módulo 1) e transações (módulo 3). Seria necessário implementar esta interface, ou então criar uma interface multiusuário para o sistema. No primeiro caso, talvez seja necessário conhecer o código interno do SGBD. No segundo, qualquer aplicação de usuário deve incluir chamada para o sistema, que com isto teria que considerar fatores como concorrência, serialização e outros.

O sistema foi implementado em banco de dados centralizado. Seria conveniente testar sua implementação em um sistema distribuido, para melhor avaliar as vantagens e desvantagens que apresenta.

As extensões propostas nos parágrafos anteriores se referem a implementação. Há, além disto, vários outros pontos em aberto que diz respeito a projeto de sistemas deste tipo. Entre outros, é necessário realizar análise de diferimento de atualização sobre

instantâneos gerados a partir de operações mais gerais (por exemplo, junções ou diferença). Além disto, talvez seja possível simplificar as estruturas de dados propostas nos casos de processamento de instantâneos independentes sobre uma mesma relação.

APÊNDICE. PROCEDIMENTOS DE MANIPULAÇÃO DO LOG.

1.INSERÇÃO

INSERÇÃO(REGISTRO: TIPO_LOG, LOG:LISTA_CIRC_CAB, LOG_ATUAL: LISTA_CIRC_CAB);

ERROS

1- Log cheio (detectado quando mensagem é recebida indicando que lista livre esta vazia)

OBS: Nesse momento o log deve ser "transformado" em transações com comandos para SGBD e os seus registros liberados novamente para a lista livre.

2- Se o pedido for de eliminação de uma tupla não existente.

3- Se o pedido for de uma tupla existente e já existe pedido anterior de eliminação.

4- Se o pedido for de inserção de uma tupla já existente.

5- Se o pedido for de alteração e tupla não existe.

OBS: Os erros 2,3,4 e 5 são detectados através do índice de chaves.

MODIFICA

LOG para LOG_ATUAL (O log passa a ter um novo registro indicando o novo pedido de atualização)

REQUER

-Que os dados sejam do tipo especificado para o registro.

- É necessário verificar para qualquer pedido de atualização, no índice de chaves, se já existe algum pedido anterior de atualização, pois dependendo da operação não há necessidade de inserir um novo registro, isto é, se a tupla a ser atualizada já está registrada no log e também para verificar a tentativa de operações inválidas.

EFEITO

1-Se não houver erro e a chave ainda não esta no índice, insira nó no índice,

2- Se não existe pedido anterior de atualização então insira o novo registro no log de atualizações

3-senão ALTERAÇÃO (chamada de procedimento).

2-REMOÇÃO.

REMOÇÃO(REGISTRO:TIPO_LOG;LOG:LISTA_CIRC_CAB;LOG_ATUAL:LISTA_CIRC_CAB);

ERROS

1-Log vazio (só existe a cabeça de lista)

MODIFICA

LOG para LOG_ATUAL (o log passa a ter um registro a menos nos casos de consulta ou em que existe pedido anterior de inserção seguido de pedido de eliminação; ou todos os registros são devolvidos para a lista livre quando finda o pré-processamento).

REQUER

Que existam atualizações pendentes

EFEITO

Se existe pedido anterior de atualização, então atualize os

ponteiros do log, devolva o registro para a lista livre, senão enquanto existir registro devolva para a lista livre. Os dados do registro eliminado são passados, quando cabível, ao módulo de geração de transações para o SGBD.

3-ALTERAÇÃO

ALTERAÇÃO(REGISTRO:TIPO_LOG,LOG:LISTA_CIRC_CAB,LOG_ATUAL:LISTA_CIRC_CAB);

ERROS

1- Registro não existe

REQUER

Dados consistentes

EFEITO

Se não existe erro então acrescente a nova atualização do registro correspondente do log.

BIBLIOGRAFIA

[ABB 1988] ABBOT, K., Mc CARTHY, D. In a replication-transparent distributed SGBD. Procedings 14 VLDB, pgs. 195 - 205, 1988.

[ANS 1975] ANSI/X3/SPARK, Study Group on Data Base Management Systems. Artigo provisório. FDT (ACM SIGMOD bulletin)7, No 2 (1975).

[BER 1984] BERNSTEIN, P.A. e GOODMAN, N. An algorthm in concurrency control and recovery in replicated distributed databases. ACM TODS, 9(4), 1984, pgs. 596 - 616.

[BLA 1986] BLAKELEY, J. A.; LARSON, P.A. E TOMPA, F.W. Efficiently Updating Materialized Views, Proc. of SIGMOD 1986, pgs. 61-71.

[BLA 1987] BLAKELEY, J. A. "Updating Materializad Database Views". Tese de doutorado, relatório técnico CS-87.32, University of Waterloo, 1987.

[BUN 1979] BUNEMAN, O.P. E CLEMON E.K. " Efficiently Monitoring Relational Databases", ACM TRANSACTIONS ON BASE SYSTEMS. Vol.4 No. 3 september 1979.

[CAR 1988] CAREY, M. J., LIVANY, M. Distributed Concurrency Control Performance: A study of algorithms, distribution and replication. Procedings VLDB 1988, pgs. 13 - 25.

[CASA 1988] CASANOVA, M.A. TUCHERMAN, L. E FURTADO, A.L. "A Monitor Enforcing Referencial Integrity". Anais 3o. SBBD, 1988, pgs. 1- 16.

[CEL 1984] CELIS, L.E.L. "Deferring Updates Without Delayed Integrity Checking", University of Waterloo, relatório não publicado.

[CER 1985] CERI, S. E PELEGATTI, G. Distributed Databases Principles & Systems. McGrawHill 1985.

[COM 1979] COER, D. The Ubiquitous B - Tree. ACM Computing Surveys vol 11(2), 1979, pgs. 121-138

[COO 1984] COOPER, E.C. "Replicated Procedure Call", 3o. PODC Confrence Proceedings. ACM 1984.

[COP 1988] COPELAND, G., ALEXANDER, W., BOUGHTER, E e KELLER, T. Data placement in BUBBA. Proceedings SIGMOD'88, pgs. 99 - 108, 1988.

[DAN 1983] DANIELS, D. E SPECTOR, A.Z. "An Algorithm For Replicated Directories", 2o. PODC Conference Procedings. ACM 1983. pgs. 24-43.

[DAT 1986] DATE, C. "An Introduction to Database Systems". 4o. edição. Addison-Wesley, 1986.

[DAY 1982] DAYAL, U. E BERNSTEIN, P.A. "On The Corre Translation of Update Operations on Relational Views", ACM TOD 8(3), pgs. 381-416,1982.

[EAG 1983] EAGER, D. e SEVCIK, K. "Achieving Robustness in

of ADMS +-: A workstation mainframe integrated architecture for DBMS". Procedings XII VLDB, pgs. 355-364.

[SEV 1976] SEVERANCE, D.G. E LOHMAN, G. "Differential files: Their aplication to the Maintenance of Large Databases". ACM TODS 1(3), pgs. 256-267.

[TUC 1988] TUCHERMAN, L. E FURTADO, A.L. "Update oriented database structures". Procedings 2o. EDBS conference, 1988, pg.10.