Uma Análise de Índices em Bancos de Dados Orientados a Objetos

Eduardo Ogasawara e Marta Mattoso

Relatório Técnico ES- 497/99-abril

Programa de Engenharia de Sistemas e Computação

COPPE/UFRJ

Uma Análise de Índices em Bancos de Dados Orientados a Objetos

1 - Introdução

Assim como no modelo relacional, os índices são componentes essenciais nos sistemas de banco de dados orientados a objetos para melhor apoiar o processamento de consultas. Em particular os índices devem acelerar a varredura de objetos que satisfaçam um determinado predicado de seleção. Bertino e Foscoli [Be Fo 95] classificam as técnicas de indexação para os bancos de dados orientados a objetos como *comportamentais* e *estruturais*. As técnicas de indexação *comportamentais* objetivam prover uma execução eficiente para consultas que contenham invocação de métodos. Elas são baseadas num pré-cálculo ou num "cache" de método, armazenando os resultados num índice. O maior problema desta abordagem consiste em detectar as mudanças nos objetos e invalidar o resultado de um método. Já os índices *estruturais* se baseiam em valores de atributos de objetos. Estes índices são importantes visto que muitas linguagens de consulta, inclusive a OQL [Ca Ba 97] (Object Query Language), permitem usar predicados que envolvam condições de acesso a objetos ao longo de uma ligação de referência entre eles (expressões de caminho), permitindo assim a navegação sobre o grafo de composição de objetos [Öz Bl 95]. As técnicas de indexação estrutural podem ser classificadas em técnicas que apoiam expressões de caminho [Ke Mo 94][Be Ki 89] e técnicas que apoiam consultas sobre hierarquia de classes [Ki Da 89].

Este trabalho concentra seu esforço na análise de índices estruturais. É feito um levamentamento e avaliação qualitativa das principais propostas de estruturas de indexação para o modelo orientado a objetos. Para tanto, o trabalho é organizado em cinco seções básicas. Esta primeira seção apresenta as definições básicas para as seções seguintes. A segunda seção trata dos índices para hierarquia de classes e faz uma análise comparativa entre eles. Uma exposição de índices para expressão de caminho é vista na terceira seção. Esta seção também avalia o comportamento destes índices face a diversas situações. Na quarta seção, são apresentados dois índices híbridos - que combinam as características de índices estruturais com as de índices para expressão de caminho - juntamente com um resumo comparativo entre os índices abordados ao longo do texto. Finalmente, a quinta seção apresenta as considerações finais sobre a adequação dos índices face às diversas situações de uma consulta OO.

1.1 - Estrutura Básica de Índices

Os índices apresentados neste trabalho tem uma estrutura básica de armazenamento em forma de árvore. Eventualmente alguns deles podem ser criados em forma de hash, mas pouca ênfase é realizada neste sentido. A não ser que dito explicitamente ao longo do texto, os índices são armazenados na forma de árvore B+.

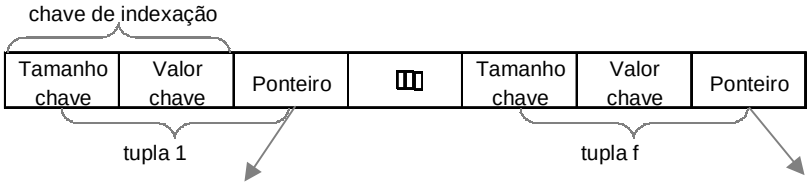

Figura 1 - Representação de um nó intermediário

Quando armazenados na forma de árvore B+, o que diferencia um índice de outro são os nós folhas. Os nós intermediários apresentam f tuplas, onde cada *tupla* é da forma *[chave de indexação, ponteiro]*. Cada *chave de indexação* tem a forma *[tamanho da chave, valor da chave]*. O tamanho da chave representa o número de bytes ocupados pelo valor da chave. Cada nó intermediário da árvore B apresenta f ponteiros de saída (f entre d e $2d$, onde d é a ordem da árvore B). O número de ponteiros que saem da raiz da árvore é de 2 a $2d$. O ponteiro de cada tupla contém o endereço físico do próximo nível do nó de indexação da árvore. Se for necessário inserir uma tupla num nó que contenha $2d$ tuplas, o nó é partido e as $2d + 1$ tuplas são distribuídas em dois nós.

1.2 – Aplicação para Estudo de Caso

Para facilitar o entendimento dos índices e auxiliar a comparação entre eles, considere a modelagem da Figura 2, que adota a representação gráfica de UML.

A aplicação tem quatro classes básicas: **Acervos, Livros, Periódicos** e **Bibliotecas**, sendo que **Livros** e **Periódicos** são especializações de **Acervos**. Neste modelo, **Acervos** representam uma classe abstrata, ou seja, não permite a existência de nenhuma instância da própria classe, há apenas a existência de membros provenientes de alguma de suas especializações, como por exemplo, **Livros**. Há, também, uma expressão de caminho entre **Bibliotecas** e **Acervos**, representado por: **Biblioteca.ListaAcervos**.

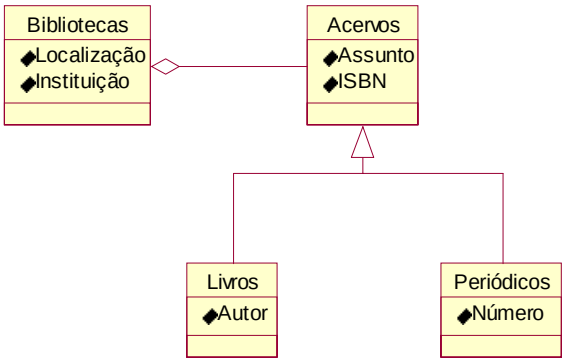

Figura 2 - Aplicação Básica

A partir deste modelo, pode-se também definir um conjunto básico de instâncias, como na Figura 3, como também diversas consultas, como as apresentadas a seguir:

Livro[L1]
0-201-59098-0
Banco de Dados
ACM Press
Kim

Livro[L2]
0-13-691643-0
Banco de Dados
Prentice-Hall
Ozsu

Livro[L3]
0-07-052182-4
Eng. de Software
McGraw-Hill
Pressman

Periódico [P1]
0-201-59098-1
Banco de Dados
ACM Press
Vol. 8; N° 4

Periódico [P2]
0-07-052182-5
Eng. de Software
ACM Press
Vol. 9; N° 1

Periódico [P3]
0-07-052182-6
Eng. de Software
ACM Press
Vol. 9; N° 2

Biblioteca [B1]
Rio de Janeiro
UFRJ
Acervo [L1]
Acervo [L2]
Acervo [P1]

Biblioteca [B2]
Niterói
UFF
Acervo [L3]
Acervo [P2]

Biblioteca [B3]
Rio de Janeiro
UERJ
Acervo [L3]
Acervo [P3]

Figura 3 - Instâncias da Aplicação

[a] Selecionar todos livros de "Banco de Dados".

Representando uma consulta envolvendo uma hierarquia de classes.

[b] Selecionar as bibliotecas do Rio de Janeiro que tenham acervos de "Banco de Dados".

A consulta 2 é predominantemente uma navegação sob uma expressão de caminho.

2 - Índices para Hierarquia de Classes

Uma das diferenças entre sistemas relacionais e sistemas orientados a objetos é a possibilidade de abstração de generalização / especialização, ou seja, classes podem ser especializadas em sub-classes. Em termos genéricos uma classe pode ter um número qualquer de sub-classes e super-classes. A maioria dos sistemas apresentam uma classe raiz, denominada *objeto*, de onde todas as outras classes são hierarquicamente descendentes. A partir deste conceito de especialização, os esquemas orientados a objetos capturam a semântica do relacionamento "é um" entre um par de classes. Isto influencia a semântica de instanciação de objetos. Um dos impactos desta influência está no escopo de uma consulta sobre uma determinada classe, que pode ser apenas sobre instâncias daquela classe ou incluir todas as instâncias de classes descendentes. Um outro grande impacto está no domínio D de um atributo de uma classe C, que pode ser apenas a classe D ou todas as sub-classes. Esta semântica de instanciação de objetos força mudanças significativas no modo no qual os sistemas de banco de dados usam índices [Ki Da 89]. A seguir são expostos cinco índices diferentes para o tratamento de hierarquia de classes.

Os exemplos apresentados nesta seção envolvem a definição de um índice hierárquico sobre o atributo **Assunto** para toda sub-árvore formada a partir de **Acervos**.

Os índices propostos, objetivam satisfazer as duas consultas abaixo:

Consulta 1. Selecionar todos acervos cujo assunto seja "Banco de Dados".

A primeira consulta busca todos os membros de **Acervos** que satisfazem o predicado **Assunto = 'Banco de Dados'**.

Consulta 2. Selecionar todos livros cujo assunto seja "Banco de Dados".

A segunda apenas busca as instâncias de **Livros** que satisfazem o predicado **Assunto = 'Banco de Dados'**.

2.1 – Índice para cada Classe - SC

O Índice para cada Classe (*Single Class Index - SC*) corresponde ao índice mais tradicional para hierarquia de classes. Esta estrutura faz com que o banco de dados mantenha um índice sobre um determinado atributo para uma única classe da hierarquia. Isto significa dizer que para apoiar a avaliação de uma consulta cujo escopo de acesso esteja na raiz ou no meio de uma hierarquia de classe, o sistema

deve manter um índice para cada classe na hierarquia e fazer as combinações correspondentes.

Como visto na Figura 4, os nós folhas do SC são constituídos pelo tamanho do registro, tamanho da chave, valor da chave, ponteiro para página de sobrecarga, número de elementos que apresentam aquela chave e a lista de identificadores de objetos (OID).

Um nó folha pode ser pequeno (menor que a página de indexação) ou grande (maior que a página de indexação). Um registro pequeno pode crescer para um registro grande ou simplesmente transpor os limites do nó folha. Existem diferentes formas para lidar com este tipo de situação. Kim e Dale [Ki Da 89] sugerem uma forma consistente para resolver o problema. Se um registro pequeno crescer a ponto de transpor os limites da página de índice, mas continuar um registro pequeno, então, a página de índice é dividida. Se, por outro lado, um registro de índice se tornar um registro grande, um nó folha é totalmente atribuído a ele, sendo que as partes que não couberem nesta página são armazenadas em páginas de sobrecarga. É para este propósito que existe área de ponteiros para páginas de sobrecarga. Se o valor deste campo for zero, pode-se assumir que o registro de índice cabe totalmente na página corrente.

Tamanho Registro	Tamanho chave	Valor chave	Ponteiro para Página de Sobrecarga	N° OIDs = n	$oid_1 \ldots oid_n$

Figura 4 - Nó folha do índice para cada classe (SC) [Ki Da 89]

Assim, para se indexar o atributo **Assunto** para hierarquia formada por **Acervo**, são necessários dois índices SC. Um para a classe **Livros** e o outro para a classe **Periódicos**. Não é necessário ter um terceiro índice SC para **Acervos**, visto que trata-se de uma classe abstrata. A Figura 5 mostra um exemplo de SC para a classe **Livros**, com valor de chave igual a **'Banco de Dados'**.

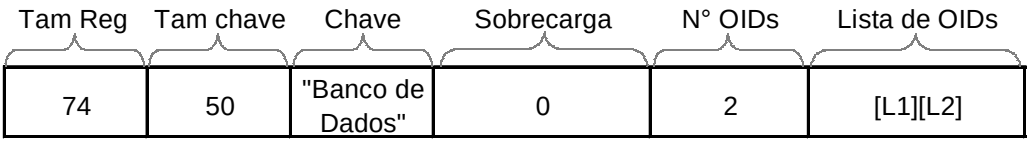

Figura 5 -Exemplo de nó folha do SC para a classe Livros

Note que para satisfazer a Consulta 1, é necessário percorrer os dois índices para buscar os OIDs que satisfaçam o predicado **Assunto = 'Banco de Dados'**. Entretanto, a Consulta 2 é facilitada, visto que é necessário apenas buscar diretamente no índice específico da classe **Livros**.

2.2 – Índice para Hierarquia de Classes - CH

O índice para hierarquia de classes (*Class Hierarchy Index - CH*) representa um índice para toda uma hierarquia de classes. A avaliação de uma consulta sobre a hierarquia ou qualquer sub-hierarquia é feita apenas sobre um único índice.

Como mostrado na Figura 6, os nós folhas de um índice para hierarquia de classes são constituídos pelo tamanho do registro, tamanho da chave, valor da chave, ponteiro para página de sobrecarga, diretório de classes (para cada classe na hierarquia que apresente aquela chave) e a lista de OIDs particionados pelas classes presentes no diretório. O diretório de classes é decomposto em pares de identificador de classe por deslocamento no registro (indicando o começo da lista de OIDs daquela classe específica).

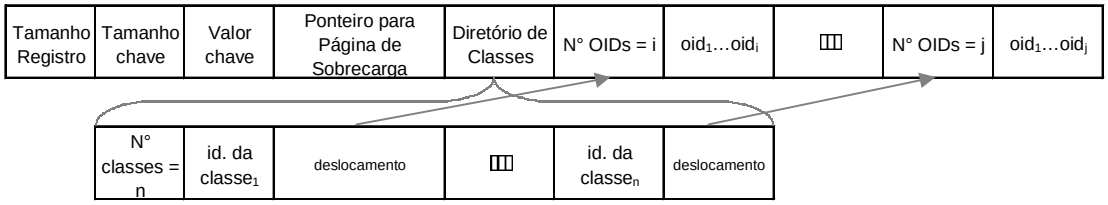

Figura 6 - Nó folha do Índice para Hierarquia de Classes (CH) [Ki Da 89]

Nesta organização, um índice para hierarquia de classes é mantido sobre um atributo numa hierarquia de classe que contenha n classes enraizadas a partir de uma classe C. No modelo da Figura 2, este índice pode ser usado para consultas que sejam

diretas a membros de Acervos. A Figura 7 mostra um nó folha para um índice CH para o atributo **Assunto** da classe **Acervos**, com valor de chave igual a **'Banco de Dados'**.

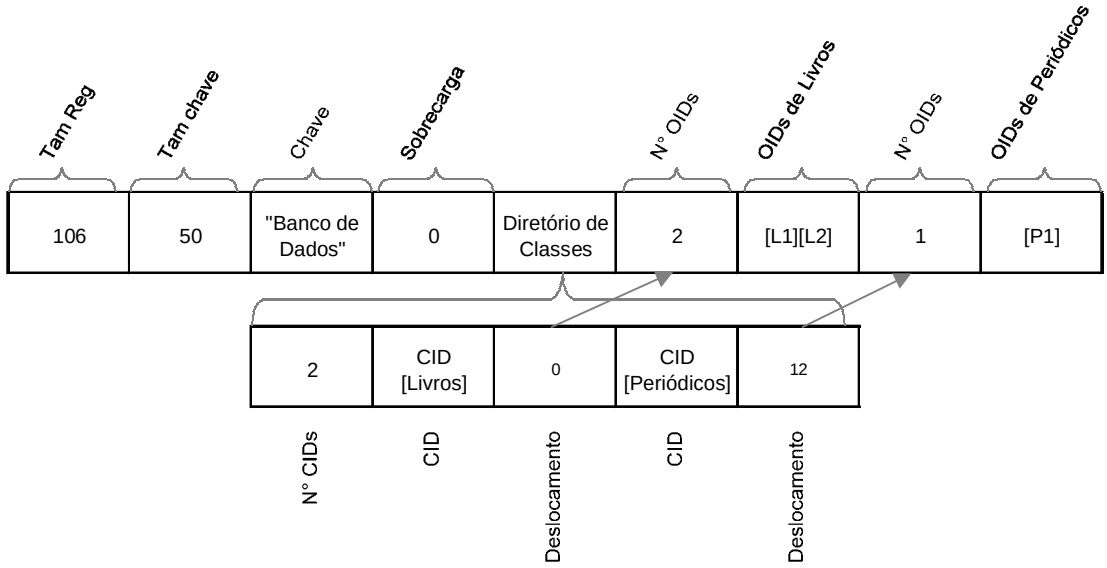

Figura 7 - Exemplo de nó folha do CH

Assim, para satisfazer a Consulta 1, é necessário percorrer apenas o CH e buscar os OIDs que satisfaçam o predicado **Assunto = 'Banco de Dados'**. Para realizar a Consulta 2 é necessário também examinar o diretório de classes, identificando a entrada que tenha o identificador da classe (CID) igual à da classe **Livros** e buscar apenas os OIDs específicos da classe.

Note que se o registro folha do índice fosse organizado simplesmente como um SC, não tendo um diretório de classes, seria necessária uma busca exaustiva sobre a lista de OIDs do resultado da busca para extrair os OIDs que não pertencessem às classes relevantes à consulta. Além disto, se uma classe fosse removida, a lista de instâncias da classe teria que ser removida do índice de hierarquia de classes. O CH facilita a remoção de qualquer classe da hierarquia.

2.3 – Árvore H Aninhada - Árvore H

Kemper e Moerkotte [Ke Mo 94] apresentam um esquema de indexação sofisticado - desenvolvido por Low, Ooi e Lu, denominado Árvore H Aninhada (Árvore H) - para combinar hierarquia de tipos num índice único. A idéia básica consiste em aninhar árvores B+, ou seja, aninhar árvores de sub-tipos dentro de uma árvore de super-tipo.

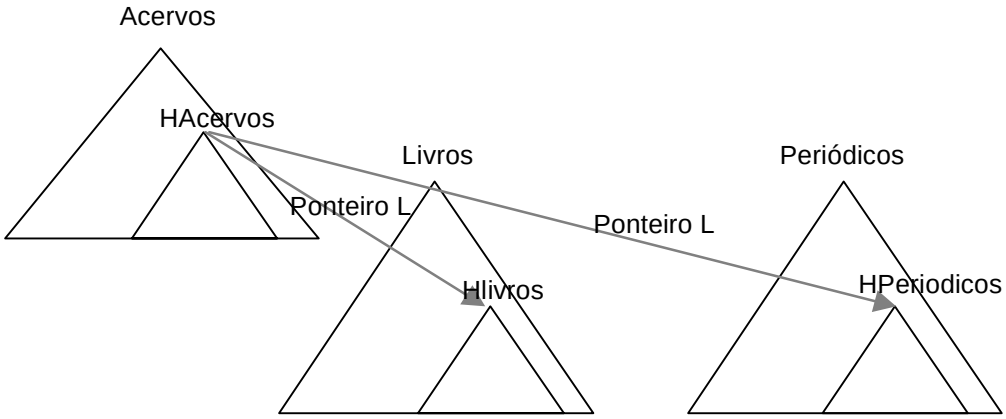

Figura 8 –Árvore H Aninhada [Ke Mo 94]

Uma varredura ao longo de uma única classe pode ser feito diretamente a partir da **Árvore H** correspondente, simplesmente ignorando os ponteiros L presentes. Uma consulta sobre toda uma hierarquia deve ser feita a partir da raiz da hierarquia (classe na qual a consulta foi iniciada), atravessando os ponteiros L para as sub-classes. Assim, para se realizar a Consulta 1, é necessário começar a busca a partir da classe ancestral **Acervos**, encontrar o valor de chave igual a **'Banco de Dados'** e percorrer todos ponteiros **L** para buscar objetos de especializações. Entretanto, para se realizar a Consulta 2, basta ir direto para a árvore de **Livros** e procurar a chave correspondente, não fazendo uso dos seus ponteiros L.

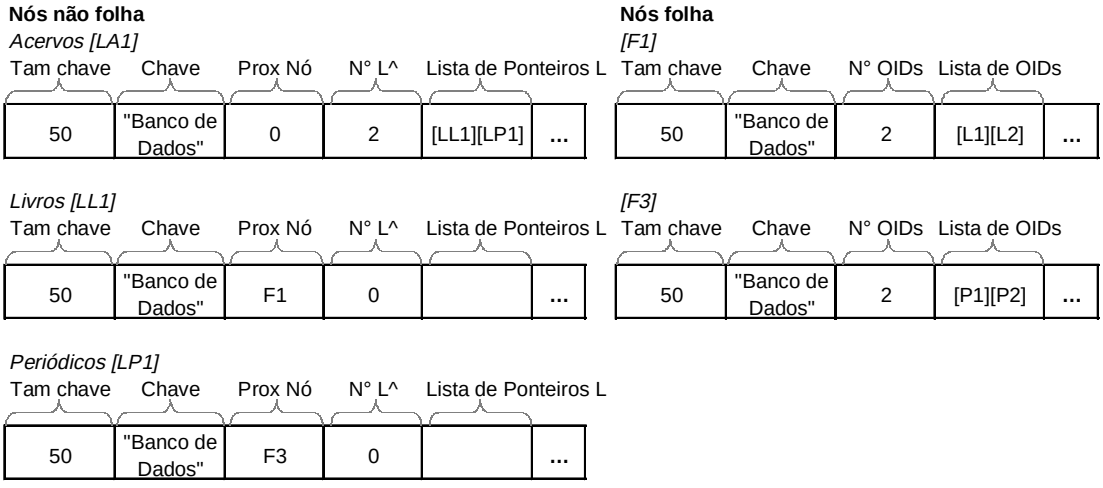

Figura 9 –Exemplo de Árvore H Aninhada

A manutenção de **Árvore H** impõe uma sobrecarga significativa em operações de atualização. Segundo Kemper e Moerkotte [Ke Mo 94], uma avaliação da penalidade para o uso desta estrutura ainda não foi investigada detalhadamente.

2.4 – Índice de Árvore para Hierarquia de Classes - χTree

Chan, Goh, Ooi, [Ch Go Oo 97] propuseram o Índice de Árvore χ para Hierarquia de Classes (χTree). A χTree combina as vantagens do CH com o SC, provendo um desempenho satisfatório tanto para as consultas voltadas em atributos como também para as consultas voltadas em classes. Para conseguir tal resultado, o índice considera tanto a dimensão das classes como também a dimensão dos dados para formar uma fragmentação do espaço de indexação. Para adicionar a grandeza de classes na dimensão de indexação é necessário definir uma linearização de tipos, ou seja, definir uma ordenação total entre as classes. Isto pode ser obtido através de uma busca pré-ordem sobre a hierarquia de classes a partir da classe ancestral (raiz da hierarquia). Como há a restrição de ordenação total, este índice não apoia sistemas que possuam herança múltipla.

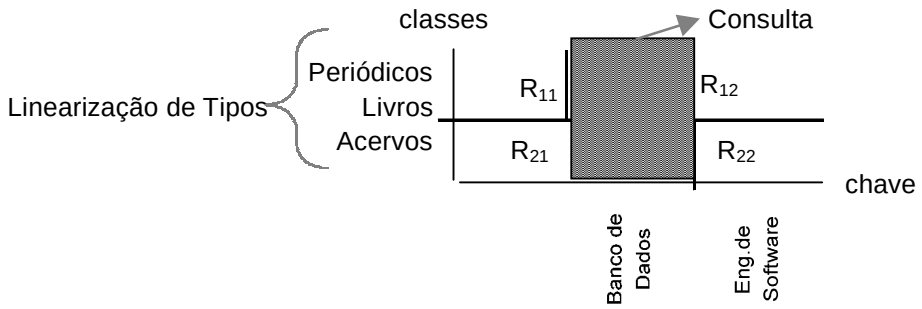

Figura 10 - χTree - [Ch Go Oo 97]

A χTree é uma estrutura de indexação balanceada muito similar a R-Tree usada em buscas espaciais. Os registros armazenados são da forma (x, y) (equivalente a pontos num espaço bi-dimensional) onde x representa a classe do objeto e y representa o atributo indexado. Os nós folha são da forma *[CID, OID, chave]*, onde *CID* representa o identificador da classe do objeto, *OID* é o identificador do objeto e *chave* é o valor da chave de indexação. Os nós não folha são da forma *[R, P]* onde *R* define uma caixa de contorno minimal e *P* e o ponteiro para o próximo nó.

Uma χTree contém um fator de utilização f (0 < f ≤ 0.5) e define, m, como o número máximo de entradas que podem ficar contidos num nó interno ou num nó folha. Assim, uma χTree tem as seguintes propriedades:

1. Cada nó tem entre $f \bullet m$ e m entradas a não ser que seja um nó folha
2. A raiz da árvore tem pelo menos duas entradas, a não ser que também seja um nó folha.
3. Se X for um nó interno com entrada (R, P). Se Y for um nó apontado por P, então:
    - Se Y for folha, então R é o menor retângulo tal que contenha todos os pontos correspondentes aos objetos em X, caso contrário
    - Se Y for um nó interno, então R é o menor retângulo tal que cubra todos os retângulos implicados pelas entradas em Y.
4. Todos os nós folhas estão no mesmo nível.

A estratégia para realizar a divisão de nós sobrecarregados é através da métrica para a medição de qualidade de divisão de um nó. Esta métrica é definida como a proximidade de acesso referente a uma determinada divisão. Suponha S_x o conjunto de registros de um nó X onde tenha ocorrido uma sobrecarga. Uma divisão, representada por [X|Y], distribui os registros de S_x entre X e um novo nó Y, particionando o conjunto S_x em dois subconjuntos: S'_x e S_y. A proximidade de acesso de uma divisão [X|Y], é definida como a probabilidade de que ambos os nós resultados da divisão X e Y, sejam acessados numa mesma consulta. Dadas duas possíveis divisões, a melhor escolha é aquela em que a probalidade de acesso da divisão seja menor. Divisões podem ser horizontais (na dimensão de classes) ou verticais (na dimensão de atributos). O algoritmo para divisão utiliza lemas para determinar as divisões horizontais ótimas de acordo com a métrica de proximidade de acesso. Após determinados também as divisões verticais ótimas, a proximidade de acesso é calculada para cada uma das divisões selecionadas. A divisão que resultar em menor proximidade de acesso é escolhida [St Na 98]. O critério para esta divisão pode ser melhor visto em [Ch Go Oo 97].

Em termos das consultas 1 e 2 usadas como exemplo, o procedimento do algoritmo é o mesmo. A diferença básica ocorre durante a execução das consultas. Neste caso, a consulta 2 gera sempre um número menor de divisões de busca pela árvore do que a consulta 1. Isto se deve à linearização das classes.

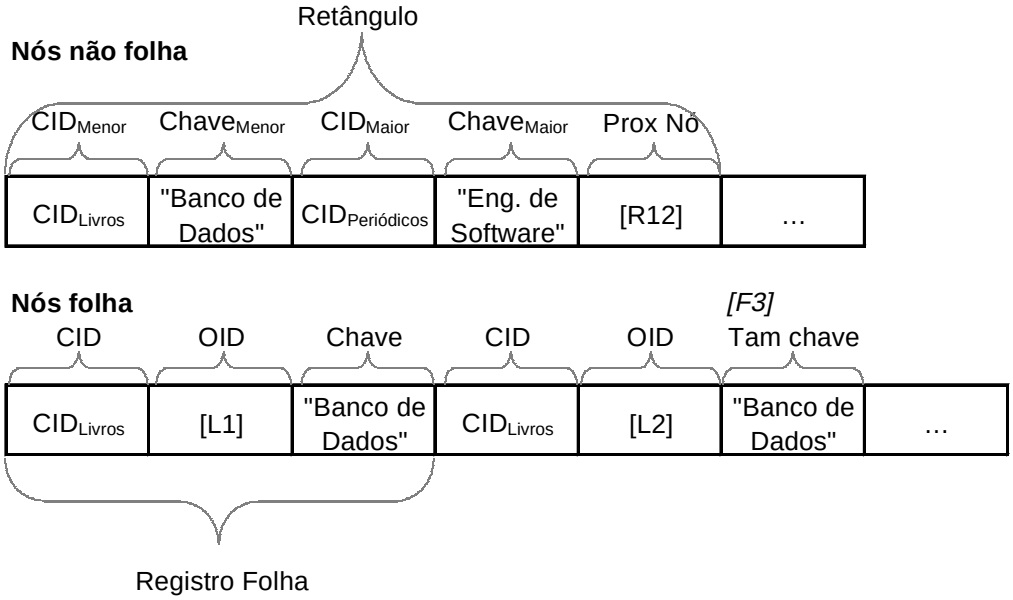

Figura 11 - Exemplo de χTree

2.5 – Índice de Tipo Multi-Chave - MT

O Índice de Tipo Multi-Chave (MT) [Mueck e Polaschek, 97] é um índice composto e representa uma estrutura alternativa para os tradicionais índices orientados em árvore B+. É um índice baseado numa *linearização ótima* de uma hierarquia de tipos. Uma linearização (ordenação total) de uma hierarquia de tipos é ótima se e somente se cada sub-hierarquia corresponder a um intervalo do domínio da ordenação total resultante.

Conceitualmente, assim como o χTree, o índice MT é uma técnica intermediária, pois mescla características de indexação por chave (representado pelo CH) com as de indexação por classes (representado por uma Árvore H).

A idéia central em torno do MT é incorporar a hierarquia de classes como atributo numa estrutura para busca de multi-atributos. Assim, a estrutura de classes é mapeada num dos domínios do índice, denominado domínio de classes. O resultado é que o índice passa a ser composto por $k + 1$ chaves, onde k representa as k propriedades do objeto a indexar de forma composta. Esta estrutura é montada numa hB-tree (cada nó interno de uma hB-Tree é organizado como uma k-d tree). Esta

estrutura para busca em multi-atributo pode ser interpretada como um conjunto no qual cada $k+1$-tupla é um elemento de um espaço geométrico de dimensão $k+1$. Baseado nesta interpretação, todas as tuplas são armazenadas num hiper-retângulo de dimensão $k+1$. Cada hiper-retângulo é fragmentado em hiper-retângulos cada vez menores, de acordo com o aumento no número de tuplas indexadas. Além disto, a hB-tree é uma estrutura que preserva a taxa de ocupação e a complexidade de estrutura de indexação para qualquer distribuição de dados. Os nós folhas representam uma lista de OIDs, sendo que esta lista pode ser precedida ou não por diretório. Se k for um valor alto, a ausência de um diretório praticamente não é percebida, devido a própria fragmentação imposta pela estrutura.

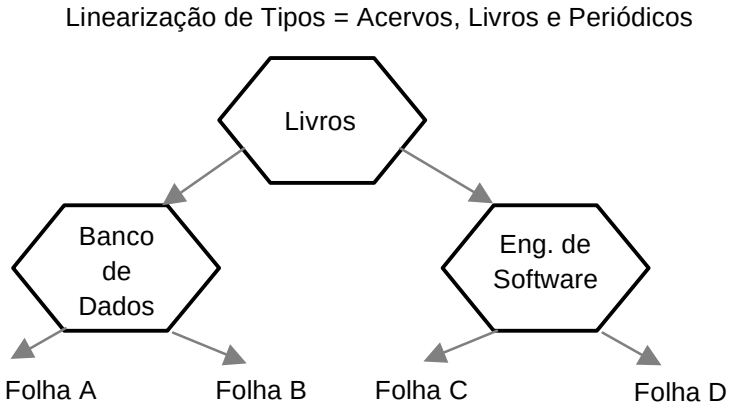

Figura 12 - Índice MT usando hB-Tree

Assim como na χTree, a Consulta 1 e a Consulta 2 usadas como exemplo, são processadas da mesma forma. Novamente a diferença básica só é percebida durante a execução das consultas. Neste caso, a Consulta 2 gera sempre um número menor de divisões de busca pela árvore que a Consulta 1 devido a linearização das classes.

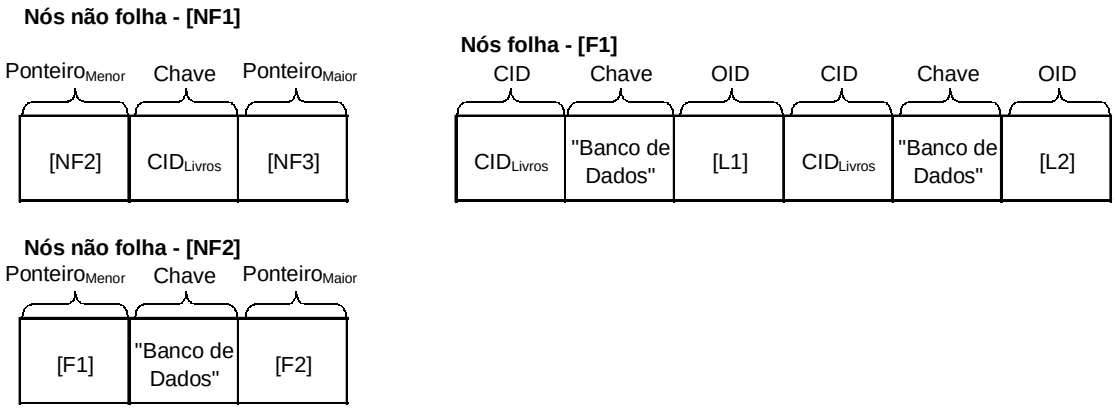

Figura 13 - Exemplo de Índice MT usando hB-Tree

2.6 - Comparação

A distribuição dos valores de chave sobre a hierarquia de classes pode ter impacto na comparação entre os índices. De acordo com Kim e Dale 89, denomina-se distribuição disjunta para o atributo indexado sobre as classes numa hierarquia a ocorrência de valores do atributo confinados a instâncias de uma única classe na hierarquia. Se os valores do atributo estão distribuídos ao longo de todas as classes na hierarquia, tem-se uma distribuição conjunta para o atributo indexado. Vale ressaltar que o conceito de distribuição conjunta/disjunta de valores é independente do conceito de distribuição uniforme/não uniforme de valores para o atributo indexado. Assim, numa distribuição disjunta de valores de atributos sobre as classes - o CH pode ser menos eficiente que o SC sobre uma classe C. Entretanto, numa distribuição conjunta - atravessar um CH pode ser muito mais eficiente que atravessar os SC referentes a cada hierarquia de classe [Ki Da 89]. Já a Árvore H, χTree e MT não são sensíveis a esta distribuição de chaves.

Analisando a sensibilidade de armazenamento dos índices em função da distribuição de chaves, verifica-se que o SC consome pouco espaço em distribuições disjuntas, mas consomem muito espaço em distribuições conjuntas, pois eles redundam muita informação dos nós não folha. Da mesma forma, as Árvore H também são sensíveis à distribuição de chaves, redundando muita informação nas distribuições conjuntas. Já o CH, χTree e MT não são sensíveis, em termos de armazenamento, à distribuição de chaves.

Em termos de consultas sobre atributos que pertencem a diversas classes, o SC é o índice que menos as apoiam. Os demais índices, por razão de construção tratam-as bem. Já para consultas sobre classes específicas, o CH é o que menos as apoiam. A Árvore H é uma estrutura voltada a realizar bem tanto as consultas orientadas a classes, quanto a atributos. Já χTree e MT, tem uma estrutura equilibrada, satisfazendo igualmente bem as consultas voltadas a classes e as voltadas a atributos. As consultas sobre faixa de domínio são pouco apoiadas pelo SC e CH. Já a χTree e MT as apoiam bem, devido à sua distribuição espacial.

O algoritmo para indexação de objetos em SC e CH é predominantemente o de uma árvore B+, tendo, assim, uma complexidade bem baixa. O MT também é simples, por ter um comportamento de uma KD-Tree, embora restrito a uma linearização de tipos. A Árvore H é um pouco mais complexa, pois requer a manutenção da integridade dos ponteiros L que ligam as sub-árvores de hierarquia descendente. Já a χTree, além de requerer a linearização de tipos, tem também um algoritmo complexo para a realização da divisão de retângulos de sua R-Tree, que faz uso do conceito de admissibilidade e proximidade de acesso.

Em termos de atualizações e remoções é que as estruturas espaciais começam a deixar a desejar. Enquanto o SC e CH apoiam trivialmente estas operações, a Árvore H tem que constantemente rever a integridade dos ponteiros L, o que torna estas operações custosas. Já a χTree e MT não apoiam estas operações, o que impossibilita fazer uso destes índices na maioria das aplicações, pois mudanças em atributos de objetos são operações bastante comuns. Assim, uma mudança como a remoção de uma classe numa SC e CH quase não é sentida, mas numa Árvore H, χTree e MT, esta operação requer uma reindexação total.

Índice **Critério**	**SC**	**CH**	**Árvore H**	**χTree**	**MT**
Sensibilidade a distribuição de chaves - desempenho	Sim	Sim	Não	Não	Não
Sensibilidade a distribuição de chaves - armazenamento	Sim	Não	Sim	Não	Não
Apoio a consultas sobre atributo	Baixo	Alto	Alto	Médio	Médio
Apoio a consultas sobre classes	Alto	Baixo	Alto	Médio	Médio
Apoio a consultas Sobre faixa de domínio	Baixo	Baixo	Médio	Alto	Alto
Complexidade em Inserções	Simples	Simples	Média	Alta	Simples

Complexidade em Atualizações	Simples	Simples	Complexa	Não apoia	Não apoia
Complexidade em Remoções	Simples	Simples	Complexa	Não apoia	Não apoia
Apoio a Mudanças na Hierarquia de Classes	Sim	Sim	Não	Não	Não
Apoio a Estruturação Em Hash	Sim	Sim	Não	Não	Não

Tabela 1 - Tabela Comparativa entre os índices hierárquicos

3 - Índices para Expressões de Caminho

As expressões de caminho são recursos poderosos na formulação de consultas. Representam navegações (travessias) entre objetos seguindo relacionamentos de forma direta evitando o processamento de junções. A maior parte das técnicas de indexação que apoiam eficientemente a avaliação de expressões de caminho é baseada numa pré-computação de ligações funcionais entre objetos. Algumas destas técnicas requerem uma simples busca num único índice para avaliar uma expressão de caminho. Entretanto, os custos de atualizações podem ser bastante altos [Be Fo 95], [Ke Mo 94]. Outros métodos [Be Ki 89] requerem percorrer um número de índices igual ao número de classes a serem avaliadas numa expressão de caminho. Entretanto, os custos de atualizações destes métodos não são tão altos como no caso anterior.

Nesta parte do trabalho são expostos quatro dos principais índices existentes para expressões de caminho. Porém, para apresentá-los, faz-se uso de algumas definições básicas. Uma formalização mais rigorosa de tais definições pode ser encontrada em [Be Ki 89], [Ke Mo 94], [Be Fo 95] e [Be Fo 97].

Seja uma expressão de caminho $P = C.A_1.A_2....A_n$, de tamanho n, uma instanciação sobre P é uma seqüência o_i de objetos $(o_1o_2...o_{n+1})$, tais que o_1 é um membro da classe C e o_i é o valor do atributo A_{i-1} do objeto o_{i-1}, para $1 < i \leq n+1$. Uma instanciação é dita parcial se o_1 começar a partir de um dos objetos em $A_1.A_2....A_{n-1}$, ou seja, $(o_1o_2...o_j)$, onde $j < n+1$. Seja uma instanciação parcial $p = o_1o_2...o_j$ sobre P, p é não redundante se não existir nenhuma instanciação $p' = o_1o_2...o_k$, $n+1 \geq k > j$, tal que $o_i = o_{k-j+i}$, $1 \leq i \leq j$ [Be Ki 89].

Uma travessia normal sobre uma expressão de caminho $C.A_1.A_2....A_n$ é percorrer uma instanciação $o_i...o_j$, para $1 \leq i < j \leq n+1$, seguindo as referências naturais dos objetos

no caminho. Uma travessia inversa sobre o caminho é percorrer esta mesma instanciação de j para i [Be Ki 89].

Uma expressão de caminho que não contenha nenhum atributo multi-valorado é denominada linear. Se, existir algum atributo multi-valorado, então o caminho é orientado a conjunto [Ke Mo 94].

O exemplo básico apresentado nesta seção envolve a definição de um índice para expressão de caminho sobre **Biblioteca.ListaAcervos.Assunto** da Figura 2. Os índices propostos, objetivam satisfazer a seguinte consulta:

Consulta 3. Selecionar as bibliotecas que tenham acervos de Banco de Dados.

Esta consulta busca todas as **Bibliotecas** que tenham algum membro de **Acervos**, em **ListaAcervos**, com predicado **Assunto = "Banco de Dados"**.

A Consulta 3 pode ser classificada como uma travessia com instanciação total numa expressão de caminho orientada a conjunto.

3.1 - Índice Aninhado - NX

Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, um Índice Aninhado (NX) sobre P é definido como um conjunto de pares $[chave, oid_1...oid_i]$, onde $chave$ representa a chave de indexação, valor do atributo A_n e $oid_1...oid_i$ é o conjunto de i objetos alcançados numa travessia inversa a partir de A_n [Be Ki 89]. Assim, um NX provê uma associação direta entre o objeto final e o objeto inicial ao longo de uma expressão.

Note que ao se falar em travessia inversa, este tipo de índice não pode ser adotado em sistemas que não tenham referências inversas, a não ser que eles não apoiem atualizações ao longo da expressão.

Como visto na Figura 14, os nós folhas do NX são constituídos pelo tamanho do registro, tamanho da chave, valor da chave, número de instanciações inversas não redundantes cujo objeto terminal tenha aquela chave e pela lista ordenada de OIDs propriamente dita. Se, eventualmente, o tamanho do registro for maior que a página de armazenamento, então um diretório é mantido em seu começo. O número de entradas no diretório é igual ao número de páginas necessárias para armazenar o registro. Cada entrada do diretório contém o endereço de uma das páginas onde é armazenada o registro e o maior OID armazenado naquela página, permitindo a fácil adição e remoção de objetos no registro.

Tamanho Registro	Tamanho chave	Valor chave	N° OIDs = n	$oid_1 \ldots oid_n$

Figura 14 - Nó folha de um Índice Aninhado - NX [Be Ki 89]

Nesta organização, pode-se definir um índice NX para a expressão de caminho **Biblioteca.ListaAcervos.Assunto**, como no modelo da Figura 2. A Figura 15 mostra um NX de tamanho dois, para o chave **Assunto** com valor igual a **'Banco de Dados'**.

Tam Reg	Tam chave	Chave	N° OIDs	Lista de OIDs
70	50	"Banco de Dados"	2	[B1][B2]

Figura 15 - Exemplo de NX com valor de chave igual a "Banco de Dados"

Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, tem-se que o NX gera uma associação direta entre o atributo aninhado A_n e a classe C, requerendo apenas uma varredura num único índice. Consequentemente, o custo de avaliação de um predicado aninhado é o mesmo que se A_n fosse atributo direto da classe C.

O custo para atualizações num NX é alto. Seja o_i $(1 \leq i \leq n)$ um objeto pertencente a uma instanciação numa expressão de caminho $P = C.A_1.A_2....A_n$. o_i tem o_{i+1} como atributo na expressão. Suponha que o_i atualize seu atributo A_i de o_{i+1} para o'_{i+1}. Para atualizar o índice, deve-se determinar S^+ (conjunto dos OIDs alcançados a partir de o_{i+1} numa travessia normal[1]) e S^- (conjunto de OIDs alcançados por o'_{i+1} numa travessia normal). Se $S^+ = S^-$, então não é necessário nenhuma modificação no índice. Caso contrário, deve-se determinar o conjunto de objetos R (objetos alcançados a partir de o_i numa travessia inversa) que devem ser modificados no índice. Para cada chave existente em $S^+ - S^-$, deve-se remover os OIDs existentes em R e, para cada chave existente em $S^- - S^+$, deve-se adicionar entradas respectivas para os OIDs existentes em R [Be Ki 89]. A única exceção a esta regra ocorre quando há alterações no último atributo da expressão de caminho. Neste caso, não são necessárias travessias diretas nem inversas, bastando uma simples mudança nos valores das chaves do índice.

[1] Se a sub-expressão formada a partir de o_{i+1} for linear, então este conjunto só tem um elemento.

3.2 – Multi-Índice - MX

Dado um caminho de tamanho n, uma indexação via Multi-Índice (MX) sobre o caminho é o conjunto de n índices MX, da forma $MX_1..MX_n$, onde cada MX é um índice NX para caminhos de tamanho 1 [Be Ki 89]. A estrutura básica do MX é idêntica ao do NX, como representada na Figura 16.

Tamanho Registro	Tamanho chave	Valor chave	N° OIDs = n	$oid_1 \ldots oid_n$

Figura 16 - Nó folha de um Multi-Índice - MX [Be Ki 89]

Para satisfazer a expressão de caminho **Biblioteca.ListaAcervos.Assunto**, deve-se definir dois índices MX. O primeiro é para satisfazer a sub-expressão **Acervo.Assunto**. O segundo é para satisfazer a sub-expressão **Biblioteca.ListaAcervos**. A Figura 17 mostra dois MXs. Um com valor de chave igual a **'Banco de Dados'**, para auxiliar a primeira sub-expressão, e o outro com o valor de chave **L1** é para a segunda sub-expressão.

MX para expressão Acervos.Assunto

Tam Reg	Tam chave	Chave	N° OIDs	Lista de OIDs
78	50	"Banco de Dados"	4	[L1][L2] [P1][P2]

MX para expressão Bibliotecas.ListaAcervos

Tam Reg	Tam chave	Chave	N° OIDs	Lista de OIDs
20	4	[L1]	1	[B1]

Figura 17 - Exemplo de MX para expressão de caminho Biblioteca.ListaAcervos.Assunto

Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, o MX requer a varredura de $(n\text{-}i+1)$ índices para avaliar um predicado sobre um atributo aninhado A_n da classe C_i $(1 \leq i \leq n)$. O índice MX_n é varrido primeiramente e uma lista de OIDs é obtida. A seguir, é feito uma junção entre a lista de OIDs e o índice MX_{n-1} e assim por diante, até que o MX_i

venha a ser acessado. Em particular, para avaliar um predicado que se refere a classe C, n índices devem ser percorridos.

Uma das grandes vantagens do MX é permitir atualizações eficientes, visto que não é necessário nenhuma varredura direta ou inversa. Supondo que um objeto o_i, $1 \leq i \leq n$, uma instância da classe C_i sobre o caminho P, seja atualizado trocando seu atributo A_i de o_{i+1} para o'_{i+1}. Então, para atualizar o índice MX_i, deve-se remover o objeto o_i do conjunto de OIDs associados com o_{i+1} e adicionar o conjunto de OIDs associados a o'_{i+1}.

3.3 – Índice de Caminho - PX

Dada uma chave de indexação, um índice de caminho PX armazena todas as i instanciações parciais não redundantes que terminam com aquela chave [Be Ki 89]. Assim, um PX pode ser representado por $[chave, (oid_{11}..oid_{1n}, oid_{i1}..oid_{in})]$, onde $chave$ novamente representa a chave de indexação, ou seja, o valor do atributo A_n numa expressão de caminho $P = C.A_1.A_2....A_n$ e $(oid_{11}..oid_{1n}, oid_{i1}..oid_{in})$ representa um conjunto das i instanciações parciais de tamanho, no máximo, igual a n.

Como visto na Figura 18, os nós folhas do PX são constituídos pelo tamanho do registro, tamanho da chave, valor da chave, número de instanciações cujo objeto terminal tenha aquela chave e pela a lista de OIDs propriamente dita.

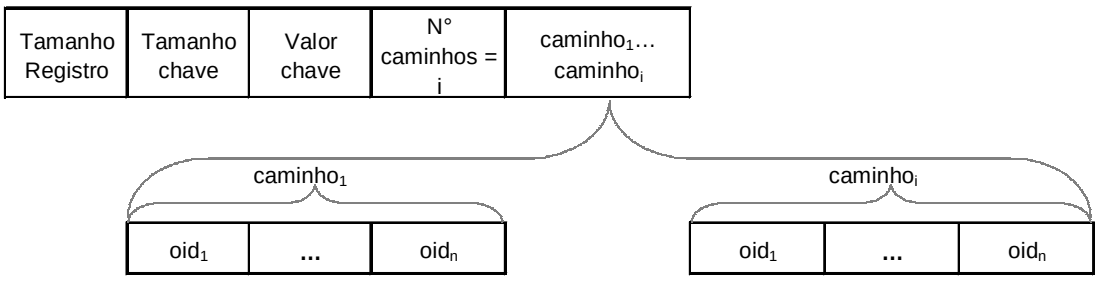

Figura 18 - Nó folha de um Índice de Caminho - PX [Be Ki 89]

Note que um PX pode ser usado para avaliar qualquer predicado com uma expressão aninhada ao longo do caminho. Se um PX e um NX forem usados para expressões de caminho de tamanho 1, então eles apresentam a mesma estrutura, tornando-se predominantemente um índice igual aos usados em sistemas relacionais [Be Ki 89].

Nesta organização, pode-se definir um índice PX para a expressão de caminho **Biblioteca.ListaAcervos.Assunto**, como no modelo da Figura 2. A Figura 19 mostra um PX para expressões de caminho de tamanho dois para a chave **Assunto** com valor igual a **'Banco de Dados'**.

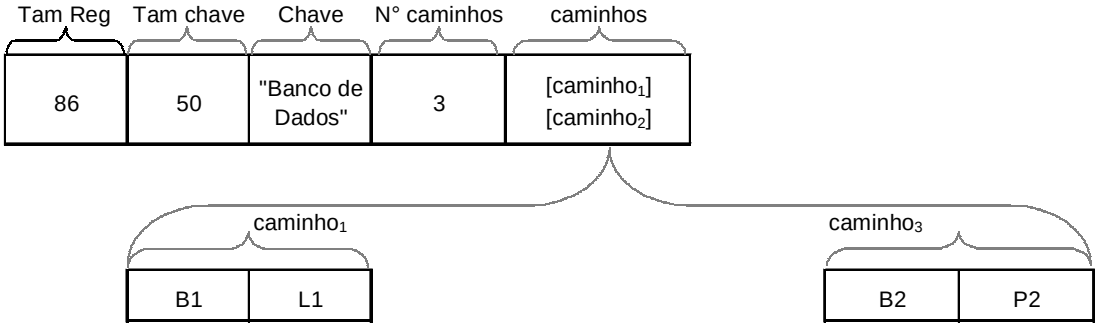

Figura 19 - Exemplo de um PX para valor de chave igual a "Banco de Dados"

O custo para atualizações de um PX também é alto. Dado o caminho $P = C.A_1.A_2....A_n$ e um objeto o_i $(1 \leq i \leq n)$ que pertença a uma instanciação, tendo o_{i+1} como atributo na expressão. Se o_i atualizar seu atributo A_i de o_{i+1} para o'_{i+1}, para atualizar o índice, deve-se determinar S^+, S^- (ver na página nº 20). Para cada chave existente em S^-, deve-se obter as expressões de caminho que passam por o_i e o_{i+1}. Nestas expressões, antes de remover estas entradas do índice, deve-se manter a parte esquerda do caminho de o_1 a o_i para que se possa compor novos registros no PX, concatenado-as com as sub-expressões $o_{i+1}...o_n$ existentes em cada uma das entradas de S^+. Uma explicação bem detalhada, com exemplos ilustrativos sobre esta operação pode ser vista em [Be Ki 89].

3.4 – Relações de Apoio à Acesso - ASR

Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, tem-se que uma Relação de Apoio à Acesso (ASR) faz uso dos $n+1$ atributos $c.o_1o_2...o_n$ para compor sua estrutura. O ASR tem uma formação muito semelhante ao PX, que, segundo Kemper e Moerkotte [Ke Mo 94], representa um caso particular de um ASR.

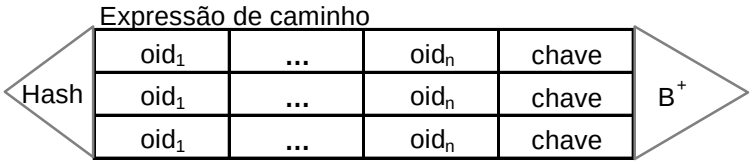

Figura 20 - Estrutura Básica de ASR - [Ke Mo 94]

Cada ASR é redundantemente armazenada em duas estruturas de índices básicas. A primeira tem como chave o atributo mais a esquerda da expressão de caminho, enquanto que a segunda é sobre o atributo mais a direita. Estas estruturas podem ser tanto Hash quanto Árvore B^+. A estrutura da esquerda apoia consultas de travessia para frente. A estrutura direita apoia travessias inversas sobre a expressão de caminho. O nó folha de um ASR é idêntico a de um PX, como mostrado na Figura 21.

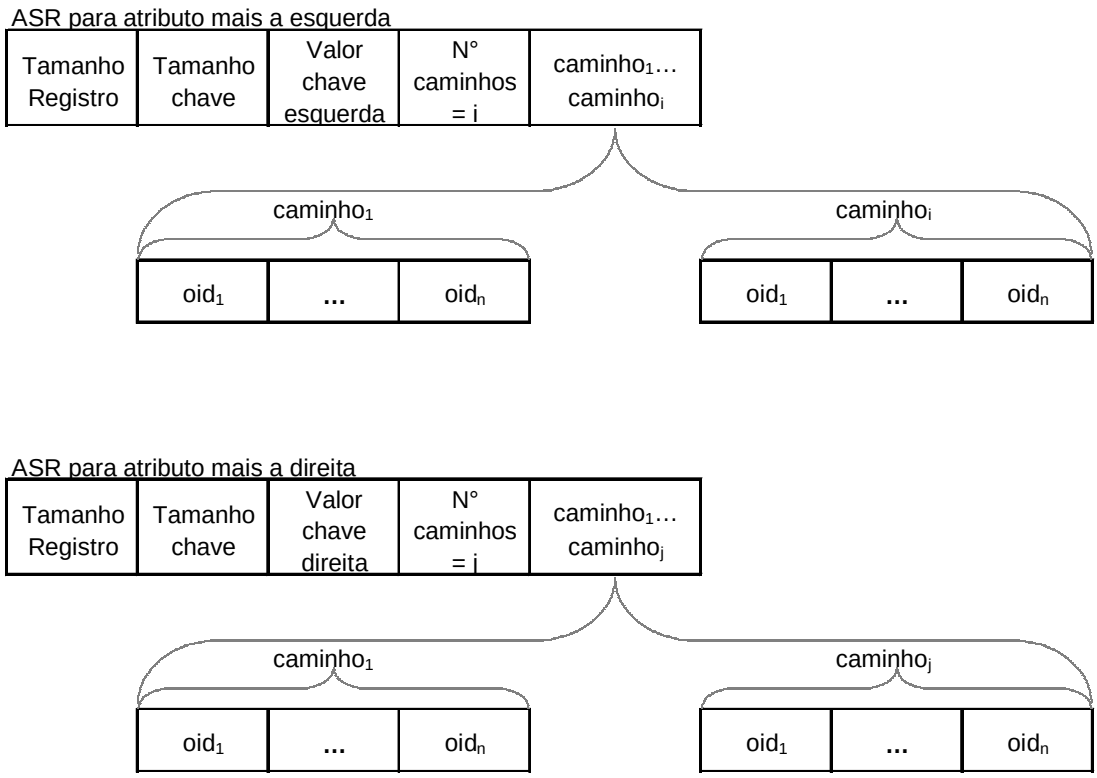

Figura 21 - Nó folha de ASR

Este esquema de índice é bem estruturado para travessias sobre caminhos tanto da esquerda para a direita, quanto da direita para a esquerda, mesmo se estas se dividirem sobre diversas estruturas ASR. Assim pode-se escolher qual dos índices que compõem o ASR deve ser usado para a consulta sobre

Biblioteca.ListaAcervos.Assunto. A Figura 22 mostra um ASR para a expressão de caminho anterior.

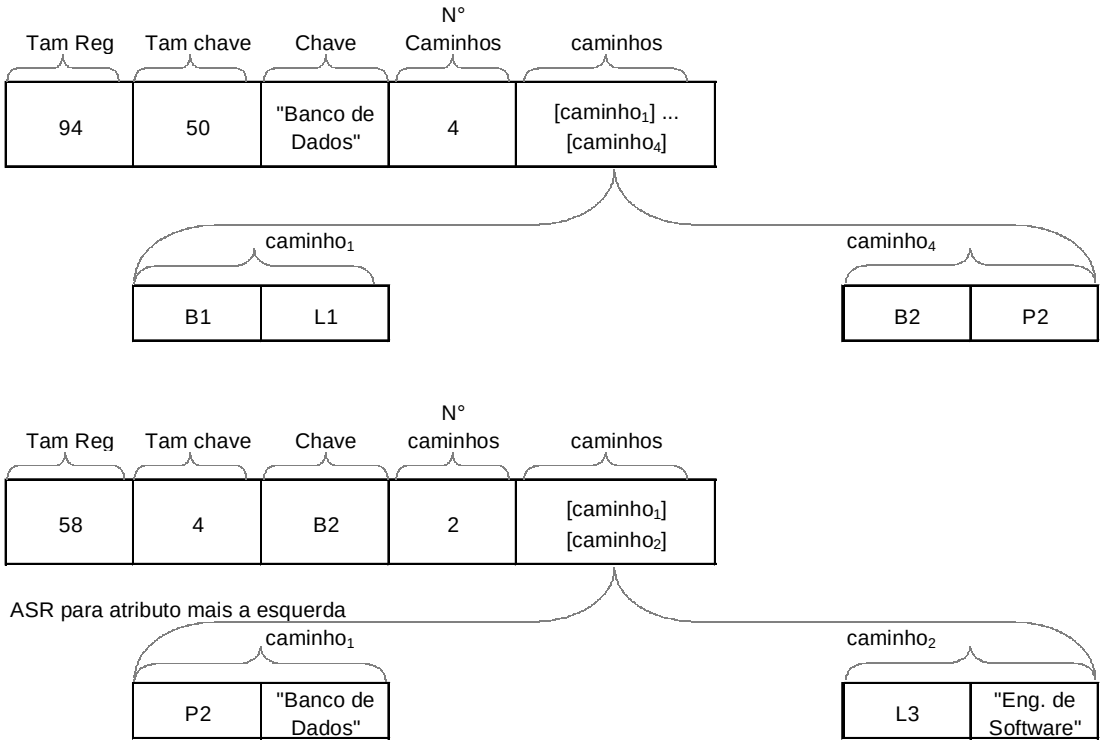

Figura 22 - Exemplo de um ASR com valor de chave esquerda igual a B2 e valor de chave direita igual a "Banco de Dados"

3.5 - Comparação

Bertino e Kim [Be Ki 89] desenvolveram um modelo de custo para realizar comparações envolvendo tamanho de índices, custo de busca e custo de atualizações apresentando resultados com os índices NX, PX e MX. Este estudo foi posteriormente aprofundado por Bertino e Foscoli [Be Fo 95].

Em termos de *tamanho de índices*, o NX - por praticamente criar uma associação direta entre o objeto final e o objeto inicial ao longo da expressão - é o que consome menos espaço. O ASR, por ser quase que equivalente a dois PX, é o índice mais custoso em termos de tamanho. A comparação entre PX e o MX é feita avaliando-se o grau de compartilhamento entre objetos. Algumas instanciações no caminho podem compartilhar algumas referências. Neste caso, os objetos referenciados são replicados

em cada uma das matrizes que implementam a instanciação. Quando não há compartilhamento de objetos, o PX tem menor custo que o MX. Entretanto, quando o grau de compartilhamento vai crescendo, a mesma informação vai sendo replicada pelo PX, aumentando o custo de armazenamento em relação ao MX, que, por sua vez, vai se aproximando do custo do NX.

A avaliação do *custo de buscas* é feita através da comparação do número de páginas trazidas à memória durante as consultas. As consultas são categorizadas em buscas por valor e buscas por faixa de domínio. Dois parâmetros muito importantes para esta comparação são k_i (número médio de instâncias da classe C_i, que contenham o mesmo valor para seu atributo A_i) e $k_i \bullet k_{i+1,}$ que representa o fator de ligação ou grau de compartilhamento entre as classes *i* e *i+1*. Para o NX e PX, o produto dos k_i tem uma representação direta no *tamanho de cada registro* nos nós folhas, que quando passa a ocupar mais de uma folha, tem um impacto grande em seu desempenho nas buscas. Já para o MX, o produto dos k_i representa o número de OIDs que devem ser avaliados durante a varredura do i-ésimo índice MX_i. O custo do MX é independente do tamanho do registro e cresce linearmente com o produto do grau de compartilhamento dos objetos.

Nas *consultas sobre faixa de domínio*, o MX leva muita desvantagem, pois há muitas junções provindas de OIDs intermediários dos índices MX_i parciais que não fazem parte do resultado final da consulta. O PX/ASR tem um resultado melhor que o MX, mas inferior ao NX. Isto se deve ao maior tamanho dos nós folhas do PX/ASR, quando comparados ao NX.

Sobre *consultas em sub-expressões* de caminho, o NX não pode ser usado eficientemente. O PX só pode ser usado em sub-expressões do tipo $A_i....A_n$, $1 < i < n$. O ASR pode ser usado em sub-expressões da forma $C....A_i$ ou $A_i....A_n$, onde $1 < i < n$. O MX pode ser usado em qualquer sub-expressão no caminho. Dependendo do tamanho da sub-expressão o MX pode ter um desempenho superior ao PX/ASR, especificamente quando a sub-expressão for de tamanho um, a superioridade do MX é indiscutível, devido ao tamanho menor dos registros nos nós folha e a não redundância de informações.

As atualizações dependem muito dos parâmetros k_i. Como no NX e PX/ASR o tamanho do registro é muito influenciado pelo produto dos k_i, a atualização pode ficar comprometida quando o tamanho do registro ultrapassa o tamanho da página. Além

disto, no NX este parâmetro fornece uma indicação muita boa de quantos objetos devem ser avaliados numa travessia inversa. O custo de atualização do ASR é praticamente o dobro do PX. A atualização no MX é bem simples. Neste caso, se o número de atualizações for intenso, então o uso do MX passa a ser essencial.

Sobre o desempenho de atualizações, tem-se que o MX é o melhor. Para expressões de caminho de tamanho dois o NX é apenas um pouco melhor que o PX. Para caminhos de tamanho três, o desempenho depende de como são distribuídas as atualizações. Se as atualizações forem predominantemente nas primeiras e segundas classes, então o NX é melhor, caso contrário o PX tem um custo menor que o NX.

A tabela 2 apresenta uma compração do custo envolvendo as diversas situações de consulta além do impacto de algumas características físicas dos índices.

Índice **Critério**	**NX**	**MX**	**PX**	**ASR**
Sensibilidade ao grau de compartilhamento – armazenamento	Médio	Baixo	Alto	Muito Alto
Sensibilidade ao grau de compartilhamento – desempenho	Baixo	Alto	Médio	Médio
Sensibilidade ao grau de tamanho do registro - desempenho	Médio	Baixo	Alto	Alto
Apoio a consultas Singulares	Alto	Médio/ Baixo	Alto/Médio	Alto/Médio
Apoio a consultas Por faixa	Alto	Baixo	Médio	Médio
Apoio a consultas sobre sub-expressões	Ausente	Alto	Médio/ Baixo	Médio
Apoio a atualizações	Baixo	Alto	Médio	Médio/ Baixo

Tabela 2 - Tabela Comparativa entre os índices para expressões de caminho

4 - Índices Híbridos

Embora a existência de índices específicos para expressões de caminho e para hierarquia de classes tenha uma grande aplicabilidade à otimização de consultas, algumas consultas envolvem tanto hierarquia de classes quanto expressões de caminho. Nestes tipos de consultas, o uso separado de índices para expressões de caminho e para hierarquia de tipos pode não ser satisfatório e nem sempre é possível aplicar seqüencialmente os dois tipos de índices. Pode-se tomar como exemplo a seguinte consulta, baseada no modelo da Figura 2:

Consulta 4. Selecionar as bibliotecas do Rio de Janeiro que tenham livros de Banco de Dados.

Esta consulta busca todas as **Bibliotecas** cujo atributo **Localização** seja "Rio de Janeiro" e que tenham algum membro de **Livros**, em **ListaAcervos**, com predicado **Assunto = "Banco de Dados"**.

Suponha que exista um índice hierárquico sobre o atributo **Assunto** para toda sub-árvore formada a partir de Acervos e um índice para expressão de caminho sobre **Biblioteca.ListaAcervos.Assunto**. Note que esta consulta pode fazer uso dos índices não convencionais de, pelo menos, três formas distintas:

1. Usando apenas o índice para hierarquia de classes, pode selecionar todos os livros de banco de dados. A seguir, se o sistema possuir referências inversas, deve-se selecionar todas bibliotecas localizadas no Rio de Janeiro.

   Esta forma de consulta é vantajosa quando o fator de seletividade de livros de banco de dados for baixo, mas requer referências inversas e, pode ser inadequada em expressões de caminho longas.

2. A segunda forma faz uso dos dois índices. Primeiro seleciona-se todos os livros de banco de dados, via índice para hierarquia de classes e, aproveitando a lista de OIDs obtida, aplica-se o índice de expressão de caminho, caso o índice permita armazenar OIDs intermediários na expressão (como o PX e MX), para selecionar as bibliotecas que tenham livros de banco de dados e verificar se elas apresentam localização no Rio de Janeiro.

   Esta forma de consulta é vantajosa quando o fator de seletividade de livros de banco de dados for muito menor que os acervos de banco de dados e quando a cardinalidade de acervos banco de dados for superior a de bibliotecas, mas não é qualquer índice para expressão de caminho que pode ser usado. Particularmente o NX, que é o índice de mais rápido acesso em expressões de caminho, não pode ser adotado.

3. A terceira forma faz uso apenas do índice para expressão de caminho. Deve-se, a partir de todas acervos de banco de dados, testá-los se eles são membros de livros, caso o índice permitir armazenar OIDs intermediários na expressão de caminho (como o PX e MX) e obter os OIDs das bibliotecas, verificando se elas apresentam localização no Rio de Janeiro.

   Esta forma é vantajosa quando o fator de seletividade de acervos de banco de dados é baixo, permitindo uma rápida checagem de classes dos OIDs. A cardinalidade de bibliotecas pode ser muito maior que a de acervos. Novamente o NX não pode ser adotado nesta consulta.

Bertino e Foscoli 95 contribuem muito no estudo de organização de índices para orientação a objetos e em apresentar duas técnicas básicas para melhor apoiar consultas que envolvam tanto expressões de caminho quanto hierarquia de classes. Para apresentá-las, faz-se uso de algumas definições adicionais. Uma formalização mais rigorosa de tais definições pode ser encontrada em [Be Fo 95] e [Be Fo 97].

Seja C uma classe, C* representa o conjunto de classes descendentes de C. Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, de tamanho n, tem-se que $classe(P) = C \cup$

{C_i | C_i é domínio do atributo A_{i-1} da classe C_{i-1}, $1 < i \leq n$} e *escopo(P)* = $\Upsilon_{C_i \in Classe(P)} C_i^*$.

Dada uma classe C' no *escopo(P)*, a posição de C' é o número inteiro i, de modo que C' pertença a hierarquia enraizada na classe C_i, onde C_i pertence a *classe(P)*.

4.1 – Multi-Índice de Herança - IMX

O Multi-Índice de Herança (IMX) é uma organização que representa um melhoramento do Multi-Índice (MX) e consiste em usar um índice para hierarquia de classe (CH) [Ki Da 89] para cada posição ao longo da expressão de caminho. Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, existe um índice em cada classe C_i em *classe(P)*. O índice em C_i associa OIDs de membros de C_i aos valores do atributo A_i [Be Fo 95].

A estrutura básica do IMX é idêntica ao do CH, como representada na Figura 23.

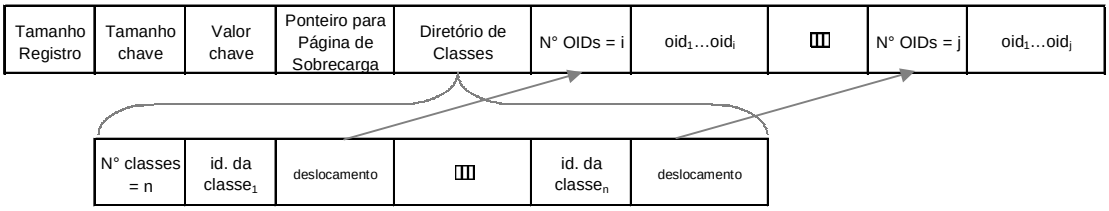

Figura 23 - Nó folha de um IMX

Para satisfazer a expressão de caminho **Biblioteca.ListaAcervos.Assunto**, deve-se definir dois índices IMX. O primeiro é para satisfazer a sub-expressão **Acervo.Assunto**. O segundo é para satisfazer a sub-expressão **Biblioteca.ListaAcervos**. A Figura 24 mostra dois IMXs. Um com valor de chave igual a **'Banco de Dados'**, para auxiliar a primeira sub-expressão, e o outro com a chave **L1** é para a segunda sub-expressão.

Salvo o melhor apoio a consultas que envolvam tanto hierarquia de classes quanto expressões de caminho, em termos de complexidade, o comportamento do IMX é muito semelhante ao MX, tendo as mesmas limitações básicas. O custo de busca num IMX é muito alto e requer a varredura dos diversos índices que o compõem. O custo de atualizações e remoções é baixo, por não requerer nenhuma travessia direta ou inversa.

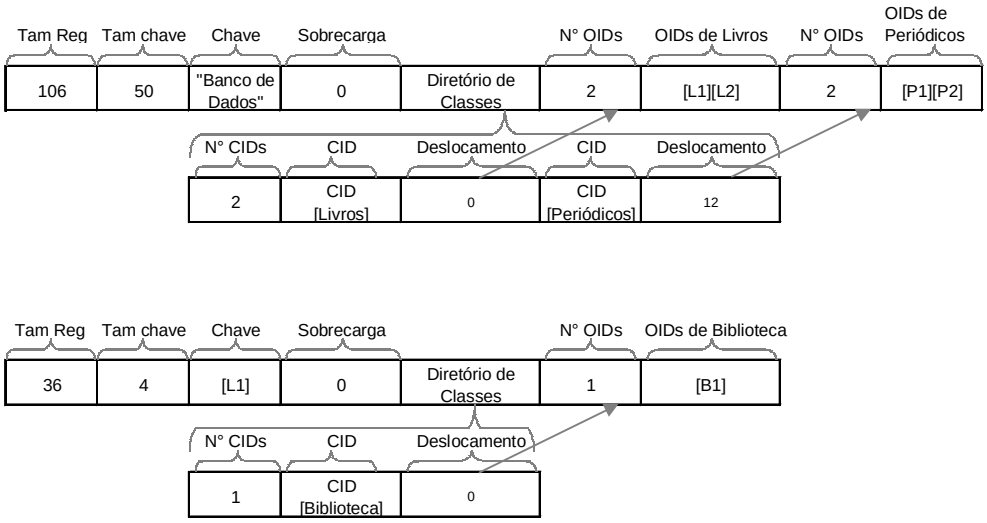

Figura 24 - Exemplo de IMX para expressão de caminho Biblioteca.ListaAcervos.Assunto

4.2 – Índice Aninhado de Herança - NIX

Em essência, o Índice Aninhado de Herança (NIX) permite que consultas contendo predicados sobre expressões de caminho e classes possam ser resolvidas numa varredura num único índice. Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, um NIX associa ao valor v do atributo A_n, OIDs de instâncias de classes no escopo de P, tendo v como atributo aninhado [Be Fo 95].

Assim como o NX, PX e ASR, o NIX apoia a rápida varredura em expressões de caminho. Entretanto, o NIX, ao contrário das organizações anteriores, não requer travessias diretas ou inversas para atualizações, visto que algumas informações adicionais são armazenadas no índice. O nó folha do NIX é denominado registro primário. Ele contém a forma exibida na Figura 25.

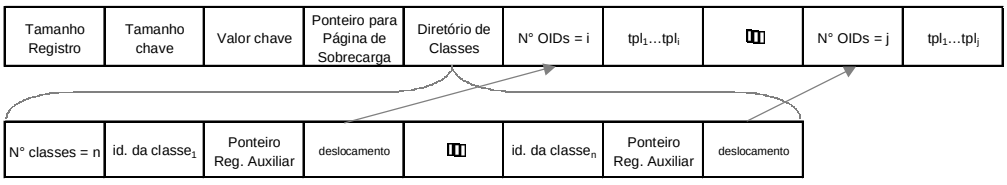

Figura 25 - Registro Primário de um NIX

Assim, o registro primário de um NIX é constituído pelo seu tamanho, tamanho da chave, valor da chave, ponteiro para página de sobrecarga, diretório de classes e, para cada classe na hierarquia que apresente aquela chave, o número de elementos

que tenham aquela chave e uma lista de tuplas *tpl*. Esta lista de tuplas é definida da seguinte forma:

1. Se A_i é um atributo simples ou $i = n$, a tupla *tpl* é apenas formada por (*OID*), onde *OID* representa o OID de uma instância daquela classe que possui o atributo aninhado A_n.
2. Se A_i é um atributo multi-valorado, a tupla é formada pelos pares (*OID, N° Filhos*) onde *OID* representa o OID de uma instância daquela classe que possua o atributo aninhado A_n e *N° Filhos* é o número de filhos daquele OID que possui o mesmo atributo aninhado A_n.

O diretório de classes contém um número de entradas igual ao número de classes tendo instâncias com aquele valor de chave como atributo indexado. Para cada classe C, tem-se uma entrada no diretório que contém:

- Identificador de classe.
- Deslocamento a partir do registro primário onde a lista de tuplas *tpl* está armazenada.
- Ponteiro para registro auxiliar onde a lista de pais é armazenada para cada instância de C_i. Um registro auxiliar é definido para cada classe, exceto para a classe raiz do caminho e seus descendentes. Um registro auxiliar consiste numa seqüência de quádruplas da seguinte forma: OID, ponteiros para registros primários, número de pais do OID e a lista de pais propriamente dita.

Existem tantas quádruplas quanto o número de instâncias de C_i. Para cada objeto o, instância de C_i, a tupla contém o identificador do objeto o, o ponteiro para os k registros primários, onde k é o número de chaves no atributo aninhado A_n de o, o número de pais de o e a lista de OIDs dos pais de o.

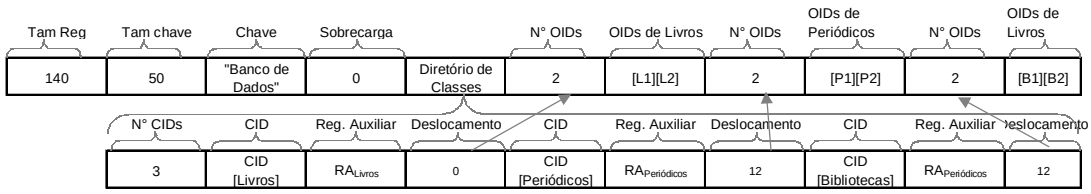

Figura 26 - Exemplo de registro primário num NIX para Biblioteca.ListaAcervos.Assunto

O registro auxiliar é indexado por uma árvore B+ baseado no OID do primeiro elemento de cada quádrupla. Consequentemente, o NIX tem uma organização que possui dois índices básicos. O primeiro índice, denominado índice primário, associa ao valor v do atributo A_n, o conjunto de OIDs de instâncias de todas as classes que tenham v como atributo aninhado A_n. O segundo índice, denominado índice auxiliar, tem OIDs como chave indexada. Ele associa o OID do objeto o, à lista de OIDs dos

pais de o. Os nós folhas do índice primário contêm ponteiros para os nós folhas do registro secundário e vice-versa. Consequentemente, o índice primário é invertido com respeito ao atributo A_n e é usado em operações de busca. O índice secundário é invertido em respeito a OIDs de instâncias de todas as classes do escopo do caminho, exceto para as de primeira posição na expressão. Basicamente o índice secundário é usado para determinar todos os registros primários onde OIDs de uma determinada instância são armazenados para eficientemente realizar operações de remoção e inserção.

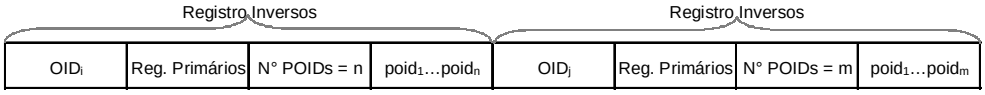

Figura 27 - Registro Auxiliar de um NIX

Assim, numa expressão de caminho $P = C.A_1.A_2....A_n$, tem-se que o NIX, assim como o NX, gera uma associação direita entre o atributo aninhado A_n e a classe C, requerendo apenas uma única varredura num índice. Consequentemente, o custo de avaliação de um predicado aninhado é o mesmo que se A_n fosse atributo direto da classe C, podendo-se ainda restringir a classe alvo da consulta.

Seja uma expressão de caminho $P = C.A_1.A_2....A_n$, e uma classe C', pertencendo ao escopo de P, tendo uma posição i. Suponha que um objeto o, instância de C' seja removido. O efeito geral da atualização do índice, é que o OID do objeto o deve ser eliminado de todo registro primário que lhe faça referência, assim como todas as instâncias que o referenciam devem ser eliminadas dos registros primários, caso elas não possuam nenhum outro filho. A manutenção do índice secundário requer que as tuplas que fazem referências a o (exceto para instâncias de classes raiz e suas subclasses) devem ser atualizadas, onde as referências a o devem ser eliminadas da lista de pais, para cada um dos filhos de o, enquanto a tupla de o deve ser removida do registro auxiliar relativo à classe C'.

Dada uma expressão de caminho $P = C.A_1.A_2....A_n$, e uma classe C', pertencendo ao escopo de P, tendo uma posição i. Suponha que um objeto o, instância de C' deva ser adicionado. Para atualizar o índice, deve-se determinar um conjunto SCH de atributos A_i de o. O índice auxiliar deve ser varrido, usando como entrada cada OID em SCH, para inclusão de o como pai dos elementos em SCH e obtenção de um conjunto de ponteiros S para os registros primários. O registro primário é varrido para cada OID em S. Uma busca no diretório de classes é avaliada para classe a C' e o OID de o é

adicionado à lista de OIDs da classe. Se A_i for multi-valorado, o número de filhos é inicializado com o seu valor adequado. Uma nova tupla é adicionada no registro auxiliar, com o OID de o como chave.

4.3 – Comparação

Os parâmetros que mais influenciam o desempenho dos índices híbridos são os valores médios de cardinalidade de ligação entre classes para cada uma das posições na expressão de caminho. Em termos de armazenamento, na maioria dos casos, o NIX tem uma taxa de ocupação muito maior que a do IMX. Entretanto, este quesito pode ser pormenorizado em função do barateamento dos dispositivos de armazenamento. Sobre os desempenhos durante a consulta, o NIX tem um desempenho superior ao IMX, embora não tenha a mesma versatilidade para uso em sub-consultas na expressão de caminho indexada. Já nas atualizações e remoções, o NIX apresenta custos altos, sendo influenciado diretamente pela topologia de referência entre os objetos existentes na expressão de caminho e pela classe alvo da consulta, tendo assim, um desempenho inferior ao do IMX, superando-o apenas em algumas situações particulares.

Em função das características de cada família de índices, seja ela pertencente aos índices hierárquicos ou aos índices para expressão de caminho, pode-se correlacionar para cada uma das estruturas de indexação, seu respectivo adequado índice híbrido substituto. A Tabela 3 exibe esta correlação.

Correlação / Índice	Família	IMX	NIX
SC	Hierárquico	**X**	
CH	Hierárquico	**X**	
NX	Expressão de Caminho		**X**
PX	Expressão de Caminho		**X**
MX	Expressão de Caminho	**X**	

Tabela 3 - Tabela de substituição por similaridade entre os índices

5 - Considerações Finais

As técnicas para indexação estrutural - índices que se baseiam em valores de atributos de objetos - podem ser classificadas em técnicas que apoiam expressões de caminho e técnicas que apoiam consultas sobre hierarquia de classes. Os índices para

hierarquia de classes possibilitam o emprego da técnica de abstração de generalização / especialização, permitindo que objetos capturem a semântica do relacionamento "é um" entre um par de classes. Os índices de hierarquia de classes podem ser orientados a valores de atributos (CH), ou seja, um único índice para toda a hierarquia de classes, como também podem ser orientados a classes, um índice para cada uma das classes existentes na hierarquia (SC) e até mesmo podem ser mistos, ou seja, tanto orientado a classes, quanto orientado a valores de atributo (Árvore H, χ-Tree, MT). Em termos de consulta, os índices espaciais são os que têm desempenho mais estáveis, mas quando a ocorrência de atualizações e remoções de objetos passa a ser uma variável relevante, a escolha do melhor índice passa a ficar restrita ao SC e CH.

A maior parte das técnicas de indexação que apoiam eficientemente a avaliação de expressões de caminho é baseada numa pré-computação de ligações funcionais entre objetos. Algumas destas técnicas requerem uma simples busca num índice para avaliar um predicado aninhado. Entretanto, os custos de atualizações podem ser bastante altos, como no NX e PX. Outros métodos, como o MX, requerem percorrer um número de índices igual ao número de classes a serem avaliadas numa expressão de caminho. Entretanto, os custos de atualizações destes métodos não são tão altos como no caso anterior.

Para tentar tirar um melhor proveito das duas famílias básicas de índices estruturais, foram desenvolvidos índices híbridos, como MIX e NIX, que melhor apoiam consultas que envolvam tanto expressões de caminho, quanto hierarquia de classes. Comparando as demais famílias de índices, com os índices híbridos, pode-se perceber que dependo das características do SGBDOO, em geral, é sempre bom adotar algum índice híbrido, visto que pode-se tirar proveito de suas vantagens, sem um aumento significativo na manutenção destes índices em comparação às demais famílias de índices.

6 - Referências

[Be Fo 95] - Bertino, E. e Foscoli, P. 1995. *Index Organizations for Object-Oriented Database Systems*. IEEE Transactions on Knowledge and Data Engeneering , Vol. 7, N° 2 (abril de 1995). Páginas 193-209.

[Be Fo 97] - Bertino, E. e Foscoli, P. 1997. *On Modeling Cost Functions for Object Oriented Databases*. IEEE Transactions on Knowledge and Data Engeneering , Vol. 9, N° 3 (maio/junho de 1997). Páginas 500-508.

[Be Ki 89] - Bertino, E. e Kim, W. 1989. *Indexing techniques for queries on nested objects*. IEEE Transactions on Knowledge and Data Engeneering , Vol. 1, N° 2. Páginas 196-214.

[Ca Ba 97] - Cattell, R., Barry, D., 1997. *Object Database Standard ODMG 2.0* - Morgan Kaufmann Publisher, Inc.

[Ch Go Oi 97] - Chan, C., Goh, C. e Ooi, B, 1997. *Indexing OODB Instances Based on Access Proximity*. IEEE International Conference on Data Engineering, abril de 1997, Birminghan, Inglaterra.

[Öz Bl 95] - Özsu M. e Blakeley, J.. 1995. *Query Processing in Object-Oriented Database Systems* in Modern Database Systems - The object Model, Interoperability and Beyond, Won Kim, Addison-Wesley, Páginas 146-174.

[Mu Pa 97] - Mueck e Polaschek, 1997. *The Multikey type Index for Persistent Object Sets*. IEEE International Conference on Data Engineering, abril de 1997, Birminghan, Inglaterra.

[Ki Da 89] - Kim, W., Kim, K. e Dale, A, 1989. *Indexing techniques for object-oriented databases* in Object Oriented Concepts, Databases, and Applications, W.Kim e F. Lochovsky, Addison-Wesley, 1989.

[Ke Mo 95] - Kemper, A. e Moerkotte, G, 1995. *Physical Object Management* in Modern Database Systems. W. Kim, Addison-Wesley, Páginas 175-202.

[St Na 98] - Stehling, R. e Nascimento, M., 1998. *Métodos de Acesso em Bancos de Dados Orientados a Objetos* em XIII Simpósio Brasileiro de Banco de Dados. Outubro de 1998, Maringá, Brasil. Páginas 369-384.