

## 1- INTRODUÇÃO

Um sistema de banco de dados não é nada mais do que um sistema de manutenção de registros por computador. O próprio banco de dados pode ser considerado uma espécie de sala de arquivo eletrônica - ou seja, um depósito de um conjunto de arquivos de dados computadorizados que oferece diversos recursos ao usuário, possibilitando-lhe a realização de várias operações, incluindo, entre outras, as seguintes:

- A adição de novos (vazios) arquivos ao banco de dados; A inserção de novos dados nos arquivos existentes;
- A recuperação de dados dos arquivos existentes; A atualização de dados nos arquivos existentes; A eliminação de dados nos arquivos existentes;
- A renovação permanente de arquivos existentes (vazios ou outros) do banco de dados.

Demonstramos um banco de dados bastante simples, que contém um único arquivo, o arquivo CELLAR que reúne informações referentes ao conteúdo de uma adega de vinhos.

### **O Arquivo CELLAR:**

BIN	WINE	PRODUCER	YEAR	BOTTLES	READY	COMMENTS
2	Chardonnay	Buena Vista	83	1	85	
3	Chardonnay	Louis Martini	81	5	84	
6	Chardonnay	Chappellet	82	4	85	Thanksgiving
11	Jo.. Riesling	Ikel	84	10	86	
12	Jo. Riesling	Buena Vista	82	1	83	Late Harvest
16	Jo. Riesling	Sattui	82	1	83	very dry
21	Fume Blanc	Ch. St. Jean	79	4	83	Napa Valley
22	Fume Blanc	Robt. Mondavi	78	2	82	
25	Wh. Burgundy	Mirassou	80	6	82	
30	Gewurztraminer	Buena Vista	80	3	82	
43	Cab. Sauvignon	Robt. Mondavi	77	12	87	
50	Pinot Noir	Mirassou	77	3	85	Harvest
51	Pinot Noir	Ch. St. Jean	78	2	86	
64	Zinfandel	Mirassou	77	9	86	Anniversary
72	Gamay	Robt. Mondavi	78	2	83	

Demonstramos abaixo um banco de dados bastante simples, que contém um único arquivo, o arquivo CELLAR que reúne informações referentes ao conteúdo de uma adega de vinhos.

```
SELECT WINE, BIN, PRODUCER FROM CELLAR
WHERE READY = 85 ;
```

Abaixo demonstração de um exemplo de uma operação de recuperação neste banco de dados, assim como os dados (mais corretamente, os resultados) devolvidos através daquela recuperação.

Resultado (impresso ou na tela):

WINE	BIN	PRODUCER
Chardonnay	2	Buena Vista
Chardonnay	6	Chappellet
Pinot Noir	50	Mirassou

Alguns outros exemplos de operações no arquivo CELLAR que, na sua maioria, são auto-explicativas. Os exemplos referentes à adição ou remoção de arquivos do banco de dados.

Inserção de novos dados:

```
INSERT INTO CELLAR
VALUES (53, 'Pinot Noir', 'Franciscan', 79, 1, 86, 'for Joan' ) ;
```

Atualização de dados existentes:

```
UPDATE CELLAR
SET BOTTLES = 4 WHERE BIN = 3 ;
```

Eliminação de dados existentes:

```
DELETE FROM CELLAR WHERE BIN = 2 ;
```

- Primeiro, por motivos óbvios, os arquivos computadorizados como o CELLAR do exemplo são chamados mais de tabelas do que de arquivos.
- Segundo, as linhas de tais tabelas podem ser consideradas registros do arquivo (às vezes chamadas explicitamente de registros lógicos, para distingui-los de outros tipos de registro . As colunas, da mesma maneira, podem ser consideradas campos destes registros lógicos.

- Terceiro, as operações SELECT, INSERT, UPDATE e DELETE demonstradas acima são, de fato, exemplos de instruções de uma linguagem de banco de dados conhecida como SQL ("Structured Query Language" - Linguagem de Consulta Estruturada). A SQL (pronunciada normalmente como "sequel") é a linguagem suportada pelos produtos de banco de dados da IBM, DB2, SQL/DS e QMF, assim como por inúmeros produtos de banco de dados de outros fabricantes.

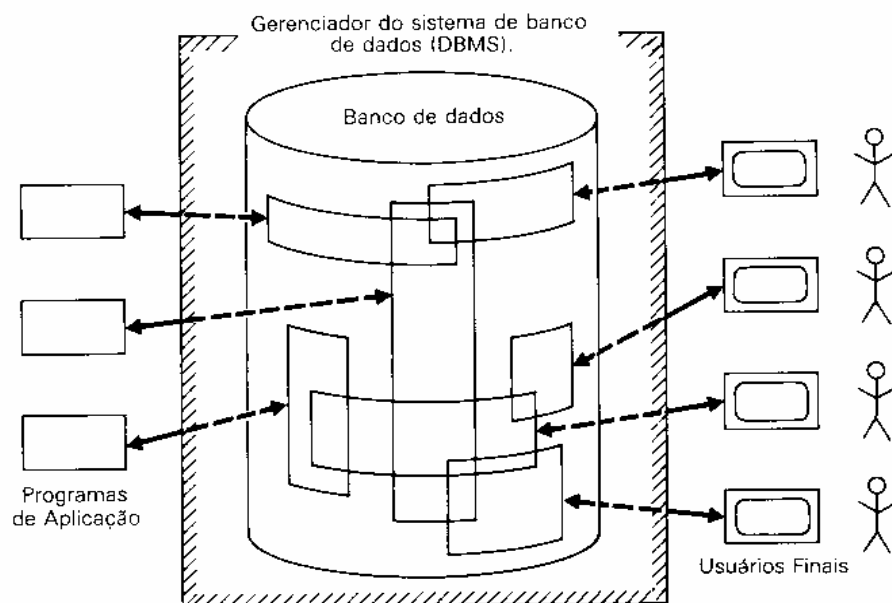
## 2 - O QUE É UM SISTEMA DE BANCO DE DADOS?

O sistema de banco de dados é basicamente um sistema de manutenção de registros por computador, ou seja, um sistema cujo objetivo global é manter as informações e torná-las disponíveis quando solicitadas. Trata-se de qualquer informação considerada como significativa ao indivíduo ou à organização servida pelo sistema - em outras palavras, que seja necessária ao processo de tomada de decisão daquele indivíduo/organização. A figura a seguir (1) abaixo mostra uma visão bastante simplificada de um sistema de banco de dados.

Esta figura pretende demonstrar que um sistema de banco de dados envolve quatro componentes principais: dados, hardware, software e usuários.

### **Dados**

Os sistemas de banco de dados agora estão disponíveis em máquinas que abrangem desde os pequenos micros até os maiores computadores de grande porte. Os recursos proporcionados por um determinado sistema são, até certo ponto, definidos pelo tamanho e pela potência da máquina básica. Os sistemas de grandes máquinas ("grandes sistemas"), em particular, tendem a ter usuários múltiplos e os das máquinas pequenas ("pequenos sistemas") a ter usuário único. Um sistema de usuário único é aquele no qual somente um único usuário pode operar num certo momento; o sistema de usuários múltiplos é aquele em que diversos usuários podem operar simultaneamente. Na realidade, a distinção é irrelevante: Um dos objetivos da maioria dos sistemas de usuários múltiplos é precisamente possibilitar a cada usuário individual comportar-se como se estivesse trabalhando com um sistema de usuário único. Os problemas especiais dos sistemas de usuários múltiplos são essencialmente internos do sistema, não visíveis ao usuário.



**FIGURA 1 – Representação simplificada de um sistema de banco de dados**

Uma outra observação preliminar: Normalmente, convém assumir, para simplificar, que a totalidade dos dados armazenados no sistema é mantida num único banco de dados; estaremos adotando esta suposição simplificada, pois não invalida substancialmente quaisquer das discussões subseqüentes. Na prática, entretanto, pode haver boas razões, mesmo num sistema pequeno, para que os dados sejam divididos em diversos bancos de dados distintos. Abordaremos algumas destas razões mais à frente.

Geralmente, pois, os dados no banco de dados - pelo menos num sistema grande - serão não só integrados como compartilhados. Estes dois aspectos, integração e compartilhamento, representam a maior vantagem dos sistemas de banco de dados de ambientes "grandes", e pelo menos a integração também pode ser significativa em ambientes "pequenos". Certamente há muitas outras vantagens (que serão discutidas mais tarde), mesmo nos ambientes ditos pequenos, mas primeiro explicaremos o que significam os termos "integrado" e "compartilhado"

Por "integrado" queremos dizer que o banco de dados pode ser imaginado como a unificação de diversos arquivos de dados que, de outra forma, seriam distintos, eliminando-se total ou parcialmente qualquer redundância entre os mesmos. Por exemplo, um certo banco de dados poderia conter tanto registros de FUNCIONÁRIOS, com nome, endereço, departamento, salário etc., como registro de INSCRIÇÃO, representando a inscrição de funcionários em cursos de treinamento. Suponhamos que, para o processo de administração de cursos, seja necessário conhecer o departamento de cada aluno inscrito. Claramente não seria preciso incluir esta informação, redundante, nos registros de INSCRIÇÃO, uma vez que ela será encontrada nos registros correspondentes aos FUNCIONÁRIOS.

Por "compartilhado" quer dizer que parcelas isoladas de dados podem ser compartilhadas por diversos usuários num banco de dados, no sentido de que todos os usuários podem ter acesso à mesma parcela de dados (e podem usá-los com finalidades diferentes). Como já mencionado, diferentes usuários podem, inclusive, ter acesso às mesmas partes de dados no mesmo momento ("acesso concorrente"). Tal compartilhamento (concorrente ou outro) é, em parte, consequência do fato de que o banco de dados é integrado. No exemplo FUNCIONÁRIO/INSCRIÇÃO acima, a informação sobre departamento nos registros FUNCIONÁRIO seria compartilhada por usuários do Departamento do Pessoal e usuários do Departamento de Educação e, como sugerido anteriormente, os dois departamentos, estariam utilizando as informações para propósitos diferentes.

Outra consequência do mesmo fato (de que o banco de dados é integrado) é que qualquer usuário, em geral, só estará interessado em um subconjunto do banco de dados total; ademais, os subconjuntos de diferentes usuários irão sobrepor-se de muitas maneiras diferentes. Em outras palavras, um determinado banco de dados será percebido por usuários diferentes de várias formas distintas. De fato, mesmo quando dois usuários compartilham o mesmo subconjunto do banco de dados, as visões do mesmo podem diferir consideravelmente a nível dos detalhes.

## **Hardware**

O Hardware compõe-se dos volumes de memória secundária – discos de cabeça móvel – nos quais reside o banco de dados, juntamente com os dispositivos associados de entrada/saída (unidades de disco, nos casos de discos de cabeça móvel), dispositivos de controle, canais de entrada/saída, e assim por diante. Este livro não se detém muito nos aspectos de hardware do sistema, pelas seguintes razões: primeiramente, esses aspectos formam, por si mesmos, um tópico maior; segundo, os problemas encontrados nesta área não são peculiares aos sistemas de bancos de dados; terceiro, estes problemas estão extensamente investigados e documentados.

## **Software**

Entre o banco de dados físico (isto é, os dados armazenados) e os usuários do sistema encontra-se o software, o gerenciador do banco de dados (o gerenciador DB) ou, mais comumente, sistema gerenciador de banco de dados (DBMS - [conforme iniciais em inglês - N.T.]). Todas as solicitações dos usuários de acesso ao banco de dados são manipuladas pelo DBMS; os recursos esboçados na Seção I.1 referentes à criação de arquivos (ou tabelas), inserção de dados, recuperação de dados etc. são todos proporcionados pelo DBMS. Outra

função do DBMS é, pois, isolar os usuários do banco de dados dos detalhes a nível de hardware (como os sistemas de linguagens de programação protegem os programadores de aplicação dos detalhes a nível de hardware). Em outras palavras, o DBMS faz com que os usuários tenham uma visão do banco de dados acima do nível do hardware, e suporta as operações do usuário (como as operações em SQL) que são expressas em termos daquela visão a nível mais elevado. Discutiremos esta e outras funções do DBMS posteriormente.

## **Usuários**

Consideramos três grandes classes de usuários. Primeiramente, três grandes classes de usuários, Primeiramente temos o programador de aplicações, responsável pela definição dos programas de aplicação que utilizam o banco de dados, caracteristicamente em linguagem como COBOL ou PL/I ou outra linguagem mais moderna; como APL ou Pascal. Estes programas operam com os dados de todas as formas usuais: recuperação de informações, criação de novas informações, anulação ou alteração de informações existentes. Todas estas funções são executadas pela emissão de solicitações apropriadas ao DBMS. Os programas em si podem ser de aplicações convencionais em lotes ou (cada vez mais) aplicações on-line, cuja função é suportar um usuário final (vide abaixo) que se comunica com o banco de dados a partir de um terminal on-line. A segunda classe de usuários, então, é o usuário-final, que interage com o sistema a partir de um terminal on-line. Um certo usuário final pode ter acesso ao banco de dados por meio de uma das aplicações on-line, definidas para o mesmo e mencionadas no parágrafo anterior, ou usar a interface fornecida como parte integrante do sistema. Tais interfaces também são fornecidas através de aplicações on-line, mas essas aplicações são embutidas e não definidas para o usuário. A maioria dos sistemas fornece pelo menos uma aplicação embutida, a saber, um processador de linguagem de consulta interativo, pelo qual o usuário é capaz de emitir comandos ou instruções de alto nível (como SELECT, INSERT etc.) ao DBMS. A linguagem SQL, já mencionada diversas vezes, é considerada um exemplo típico de linguagem de consulta de banco de dados.

### **Observa-**

**ção:** Muitos sistemas também proporcionam interfaces embutidas adicionais, nas quais os usuários não precisam emitir expressamente os comandos, como SELECT; operam (por exemplo) escolhendo itens do menu ou preenchendo-os em formulários. Tais interfaces acionadas por menus ou formulários são normalmente mais fáceis para as pessoas sem treinamento formal em processamento de dados. As interfaces acionadas por comando (i.e., linguagens de consulta), ao contrário, necessitam de uma certa experiência em processamento de dados, embora não muito grande (obviamente não tanto quanto seria necessário para definir um programa de aplicação em COBOL ou PL/I). Também neste caso, as interfaces acionadas por comando são provavelmente mais flexíveis do que as de menu ou formulários, posto que as

linguagens de consulta proporcionam certas funções que não são suportadas por outras interfaces.

A terceira classe de usuários é o administrador do banco de dados ou DBA. Assim, completamos nossa descrição preliminar dos principais aspectos de um sistema de banco de dados. Passaremos agora a discuti-los de forma mais detalhada.

### 3 – DADOS OPERACIONAIS

Um dos primeiros textos didáticos sobre o assunto, de Engles [1.10], refere-se aos dados de banco de dados como "dados operacionais", distinguindo-os de dados de entrada, dados de saída e de outros tipos de dados. Damos abaixo uma versão modificada da definição original de Engles acerca de *banco de dados*:

- Um banco de dados é uma coleção de dados operacionais armazenados, usados pelos sistemas de aplicações de uma empresa específica.

Esta definição requer uma explicação. Primeiramente, "empresa" é apenas um termo genérico e conveniente para designar uma organização comercial, científica, técnica ou de outra natureza que seja razoavelmente independente. Uma empresa pode ser um único indivíduo (com um pequeno banco de dados particular), ou uma organização de grande porte ou similar (com um grande banco de dados compartilhado), ou algo entre esses extremos. Exemplos de empresas:

- fábricas;
- bancos;
- hospitais;
- universidades;
- departamentos governamentais.

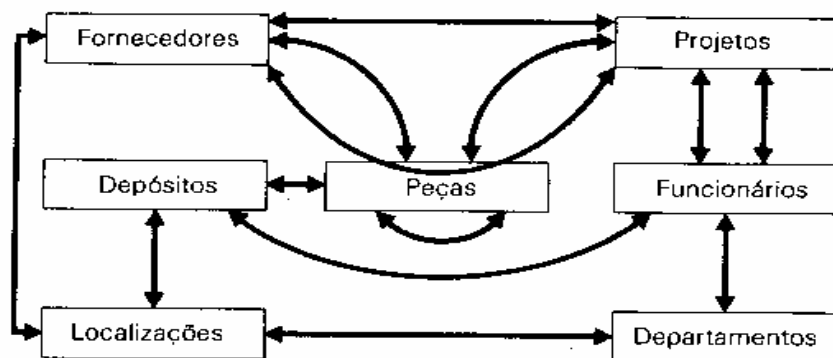
Qualquer empresa deve, necessariamente, manter uma certa quantidade de dados acerca de suas operações. São estes os "dados operacionais". Os dados operacionais das empresas acima provavelmente incluiriam o seguinte:

- dados sobre os produtos;
- dados contábeis;
- dados sobre os pacientes;

- dados sobre os estudantes;
- dados de planejamento.

Os "dados operacionais", como mencionado, não incluem os dados de entrada e saída, seqüência de trabalhos a realizar, resultados temporários ou, ainda, qualquer informação puramente transitória. Os "dados de entrada" representam informações que entram no sistema pela primeira vez (a partir do teclado do terminal, leitor de cartões ou dispositivo similar); tais informações podem provocar a necessidade de uma alteração nos dados operacionais (para que possam tornar-se parte dos dados operacionais) mas, inicialmente, não fazem parte do banco de dados em si. Da mesma forma, os "dados de saída" são mensagens e resultados que procedem do sistema (impressas ou projetadas de outra maneira numa tela de terminal); novamente, podem derivar dos dados operacionais, sem que sejam consideradas, em si, como parte do banco de dados. Observações análogas aplicam-se a outras espécies de informações transitórias.

A título de ilustração do conceito de dados operacionais, vamos considerar o caso de uma fábrica. A empresa desejará guardar informações sobre os seus projetos; as peças usadas nos mesmos; os fornecedores dessas peças; os depósitos onde são estocadas; os funcionários que trabalham nos projetos; e assim por diante. São estas as entidades básicas relativas às informações a registrar (o termo "entidade" é amplamente usado nos meios de bancos de dados para denominar qualquer objeto distinto a ser representado no banco de dados). Veja a Figura 2.



**FIGURA 2 – Exemplo de dados operacionais**

É importante notar que, além das entidades básicas em si, existirão relacionamentos interligando-as. São demonstrados pelas setas de ligação. Há, por exemplo, um relacionamento entre os fornecedores e as peças: cada fornecedor fornece certas peças e, de modo inverso, cada peça é suprida por certos fornecedores (ou melhor, cada fornecedor fornece certos tipos de peças,

cada tipo de peça é suprido por certos fornecedores). Do mesmo modo, as peças são usadas em projetos, e inversamente, os projetos utilizam peças; as peças são armazenadas em depósitos e os depósitos estocam as peças, e assim sucessivamente. Observa-se que estes relacionamentos são bidirecionais - isto é, podem ocorrer em ambas as direções. O relacionamento entre funcionário e departamentos, por exemplo, pode ser utilizado para responder às duas questões seguintes:

1. Dado um funcionário, encontre o departamento correspondente;
2. Dado um departamento, encontre os funcionários correspondentes.

O ponto importante dos relacionamentos como ilustrados acima na figura 2 é que são tanto uma parte dos dados operacionais como entidades básicas. Conseqüentemente, devem estar representadas no banco de dados. Consideremos mais à frente as diversas maneiras de fazê-lo.

A Figura 2 também ilustra outros itens.

1. Embora a maioria dos relacionamentos do diagrama envolva dois tipos de entidade - isto é, sejam relacionamentos binários -, isto não significa que todos os relacionamentos sejam necessariamente binários neste sentido. Temos no exemplo um relacionamento que envolve três tipos de entidade (fornecedores, peças e projetos) - um relacionamento ternário, interpretado de forma que certos fornecedores forneçam certas peças para certos projetos. Observe-se que este relacionamento ternário "fornecedores fornecem peças para projetos" não é, em geral, o equivalente à combinação dos três relacionamentos binários, "fornecedores fornecem peças", "peças são usadas nos projetos" e "projetos são fornecidos por fornecedores". A informação, por exemplo, que

- a) Smith fornece chaves-inglesas para o projeto Manhattan nos diz mais do que a combinação
- b) Smith fornece chaves-inglesas,
- c) chaves-inglesas são utilizadas no projeto Manhattan, e
- d) o projeto Manhattan é abastecido por Smith

Não podemos (incontestavelmente!) deduzir: a) se conhecemos somente b), c) e d). Mais explicitamente, se conhecemos b), c) e d), podemos deduzir que Smith fornece chaves-inglesas para algum projeto (digamos, projeto Jz), que algum fornecedor (digamos, fornecedor Sx) fornece chaves-inglesas para o projeto Manhattan e, que Smith fornece alguma peça (digamos, peça Px) para o projeto Manhattan -, mas não podemos realmente concluir que Sx é Smith ou que Py sejam chaves-inglesas nem que Jz seja o projeto Manhattan. Conclusões errôneas como esta ilustram o que chamamos de armadilha de conexão.

2. O diagrama mostra também uma seta que envolve apenas um tipo de entidades (peças). O relacionamento indica, neste caso, que certas peças incluem outras peças como componentes

imediatos (o chamado relacionamento "lista de materiais que compõem um produto" ) - por exemplo, um parafuso é componente de uma dobradiça, que também é considerada uma peça e que, por sua vez, pode ser um componente de uma peça maior, como uma tampa. Observemos que ainda se trata de um relacionamento binário; como dois tipos de entidades que se ligam entre si (ou seja, peças e peças) são uma só e a mesma.

3. Em geral, um dado conjunto de tipos de entidades pode enlaçar-se em qualquer número de relacionamentos. No diagrama, projetos e empregados são ligados por duas setas, uma poderia representar o relacionamento "trabalha no" (o funcionário trabalha no projeto), e outra o relacionamento "é o gerente de" (o funcionário é o gerente do projeto).

Um relacionamento também pode ser considerado uma entidade. Se tomarmos nossa definição de entidade "qualquer objeto do qual desejamos registrar informações", então relacionamento certamente se encaixa na definição. Por exemplo, "a peça P4 está estocada no depósito W8" é uma entidade sobre a qual gostaríamos de registrar informações - por exemplo, a quantidade correspondente. Neste livro, pois, estaremos considerando os relacionamentos simplesmente como um tipo especial de entidade.

#### 4 - POR QUE BANCO DE DADOS?

Por que usar um sistema de banco de dados? Quais são as vantagens? A resposta depende, até certo ponto, do sistema, se este servirá a um usuário único ou múltiplos - ou, para ser mais exato, há inúmeras vantagens adicionais no caso de usuários múltiplos. Consideramos, primeiramente, o caso do usuário único. Referimo-nos novamente ao exemplo da adega de vinhos, o qual julga-se característico de um banco de dados de usuário único. Este banco de dados específico é tão pequeno e simples que as vantagens podem não ser óbvias de imediato. Imaginemos, porém, um banco de dados similar para um grande restaurante, com um estoque de talvez milhares de garrafas e com freqüentes alterações; ou uma loja de bebidas alcoólicas, com um estoque imenso e muita rotatividade. (Embora se trate de um banco de dados maior, ainda é um sistema de usuário único.).

As vantagens do sistema de banco de dados em relação aos métodos tradicionais, baseados em papéis e arquivos ficarão mais evidentes nos seguintes exemplos:

- É compacto: Não há necessidade de arquivos de papéis volumosos.
- É rápido: A máquina pode recuperar e modificar os dados muito mais rapidamente do que o ser humano. Em especial, as consultas incidentais, repentinas (como, p. ex., "Temos mais Zinfandel do que Pinot Noir?") são rapidamente respondidas, sem consultas a manuais ou pesquisas visuais, que consomem muito tempo.

- Importa em menos trabalho braçal: elimina a maior parte do tedioso trabalho manual de arquivamento. As máquinas sempre executam as tarefas mecânicas melhor do que nós.
- Tem fluxo corrente: disponibilidade de informações certas e atualizadas a qualquer momento, basta pedir.

As vantagens acima são mais significativas em ambientes de usuários múltiplos onde o banco de dados é maior e mais complexo do que o de usuário único. Há, neste caso, outra vantagem dominante, a saber: *O sistema do banco de dados proporciona* à empresa o controle centralizado de seus dados operacionais (é uma de suas propriedades mais úteis). Tal situação contrasta nitidamente com a que vemos na empresa sem sistema de banco de dados, onde cada aplicação dispõe de seus próprios arquivos - muitas vezes também suas fitas e discos particulares - de tal forma que os dados operacionais são muito dispersos, dificultando o controle sistemático.

Vamos aprofundar o conceito de controle centralizado: implica que (numa empresa com um sistema de banco de dados) exista uma pessoa identificável detendo a responsabilidade central sobre os dados operacionais. Esta pessoa é o administrador do banco de dados (DBA); por enquanto; é suficiente sabermos que o cargo requer a) um alto grau de capacitação técnica, e b) a capacidade de entender e interpretar as necessidades da empresa a nível de gerência executiva. Na prática, a função do DBA pode ser desempenhada por um grupo de pessoas, gerentes e técnicos, ao invés de apenas um indivíduo. Contudo, para simplificar, partiremos do princípio que o DBA seja na verdade uma única pessoa. É importante percebermos que a posição do DBA dentro da empresa é (ou deveria ser) de alto nível gerencial.

Descrevemos a seguir algumas das vantagens que resultam da noções de controle centralizado:

- Pode reduzir a redundância.

Nos sistemas sem banco de dados, cada aplicação possui seus próprios arquivos. Este fato costuma provocar uma redundância considerável nos dados armazenados, com o desperdício de espaço de armazenamento resultante. Por exemplo, uma aplicação de pessoal e uma aplicação de registros de educação podem ter, ambas, um arquivo contendo informações do departamento relativo aos funcionários. Como sugerido na Seção 1.2., estes dois arquivos podem ser integrados, e a redundância eliminada, se o DBA estiver a par das necessidades de dados de ambas as aplicações - isto é, se o DBA possuir o necessário controle global.

Não pretendemos sugerir que toda a redundância deva necessariamente ser eliminada. Algumas vezes, existem fortes razões, técnicas ou comerciais, para se manter cópias múltiplas do mesmo dado. Entretanto sugerimos que a redundância seja cuidadosamente controlada - isto é,

o DBMS deve ter conhecimento da mesma, e assumir a responsabilidade de "propagar as atualizações" (vide o item abaixo).

- A inconsistência pode ser evitada (até certo ponto).

Conseqüência natural do item anterior. Suponhamos que um certo fato do mundo real - digamos, o fato de que o funcionário E3 trabalha no departamento D8 - é representado por duas entradas distintas no banco de dados, e que o DBMS não tenha conhecimento desta duplicidade (i.e., a redundância não é controlada). Haverá, então, ocasiões em que as duas entradas não serão concordes - ou seja, quando apenas uma das duas entradas for atualizada. Diz-se, então, que o banco de dados é inconsistente. É óbvio que um banco de dados considerado inconsistente é capaz de fornecer informações incorretas ao usuário.

Se o fato mencionado estiver representado por uma única entrada (isto é, se a redundância for removida), obviamente tal inconsistência não poderá ocorrer. Alternativamente, se a redundância não for removida, mas controlada (tornando-se conhecida do DBMS); então ele poderá garantir que o banco de dados nunca estará inconsistente, como visto pelo usuário, assegurando que qualquer alteração feita em uma das entradas seja automaticamente aplicada na outra. Conhece-se este processo como propagação de atualização - onde (como ocorre em geral) se utiliza o termo "atualização" para cobrir todas as operações de inserção, anulação e modificação. Observemos, entretanto, que poucos sistemas comercialmente disponíveis hoje são capazes de propagar automaticamente as atualizações; isto é, a maioria dos produtos atuais não suporta a redundância controlada, exceto em poucos casos especiais.

- Pode compartilhar os dados.

O compartilhamento não significa apenas que as aplicações existentes podem compartilhar os dados do banco de dados, mas também que novas aplicações podem ser desenvolvidas para operar sobre os mesmos dados armazenados. Em outras palavras, as necessidades de dados das novas aplicações podem ser satisfeitas sem a criação de quaisquer dados adicionais armazenados.

- Pode reforçar os padrões.

O DBA, pelo controle central do banco de dados, pode assegurar que todos os padrões aplicáveis serão observados na representação dos dados. Os padrões aplicáveis podem incluir um ou todos, mencionados a seguir: padrões a nível de instalações, departamentos, indústrias,

padrões nacionais ou internacionais. Uma padronização dos formatos dos dados armazenados é especialmente interessante para facilitar o intercâmbio de dados ou a migração entre sistemas. Da mesma forma seria desejável a denominação dos dados e a padronização da documentação para facilitar o compartilhamento e a compreensão dos dados.

- Pode aplicar restrições de segurança.

O DBA, detendo toda a autoridade sobre os dados operacionais, pode assegurar a) que os únicos meios de acesso ao banco de dados sejam realizados através de certos canais e, conseqüentemente, b) pode definir os controles de segurança a adotar, sempre que for empreendido o acesso a determinados dados especiais. Pode estabelecer diferentes controles para cada tipo de acesso (recuperação, modificação, anulação etc.) e para cada parte da informação no banco de dados. Observemos, porém, que os dados, sem tais controles de segurança, podem incorrer num risco maior do que em sistema tradicional de arquivo (disperso); isto é, a natureza centralizadora do sistema de banco de dados requer, também, um bom sistema de segurança.

- Pode manter a integridade.

O problema da integridade é assegurar que os dados do banco de dados sejam corretos. A inconsistência entre duas entradas que pretendem representar o mesmo "fato" é um exemplo de falta de integridade (vide discussão no item acima); este problema, certamente, só pode ocorrer se houver redundância nos dados armazenados. Entretanto, mesmo que ela não exista, o banco de dados ainda pode conter uma informação incorreta. Por exemplo, estaria registrado que um funcionário trabalhou 400 horas na semana, em vez de 40 horas, ou que o mesmo pertence a um departamento inexistente. O controle centralizado do banco de dados ajuda a evitar tais problemas - à medida que possam ser evitados -, pois permite que o DBA defina controles de integridade a realizar sempre que for empreendida qualquer operação de atualização. (Utilizamos novamente o termo "atualização" em termos genéricos, a fim de abranger todas as operações de modificação, inserção e anulação).

Chama-se a atenção para o fato de a integridade de dados ser ainda mais importante em sistemas de banco de dados de usuários múltiplos do que nos ambientes de "arquivos particulares", precisamente porque o banco de dados é compartilhado. Sem controles apropriados, um usuário poderia atualizar o banco de dados incorretamente, gerando dados errados e, assim "infectando" outros usuários. A mencionar, ainda, que a maioria dos produtos atuais de banco de dados são um tanto fracos no suporte de controles de integridade.

- Pode equilibrar as necessidades conflitantes.

O DBA, tendo conhecimento das necessidades globais da empresa - em oposição às necessidades de um usuário individual - pode estruturar o sistema, a fim de proporcionar um serviço geral que seja "o melhor para a empresa". Por exemplo, pode-se escolher uma representação para os dados na memória, dando rápido acesso às aplicações mais importantes, em detrimento do desempenho mais fraco de determinadas aplicações.

A maioria das vantagens mencionadas acima é provavelmente bastante óbvia. Entretanto há necessidade de adicionarmos um outro item à lista, que não é tão óbvio - muito embora tenha relação com diversos itens anteriores - ou seja, a independência de dados. (Na verdade, é mais um objetivo dos sistemas de banco de dados do que necessariamente uma vantagem.) O conceito da independência de dados é tão importante que dedicaremos uma seção exclusivamente ao mesmo.

## 5 – ARQUITETURA DE UM BANCO DE DADOS

### OS TRÊS NÍVEIS DA ARQUITETURA

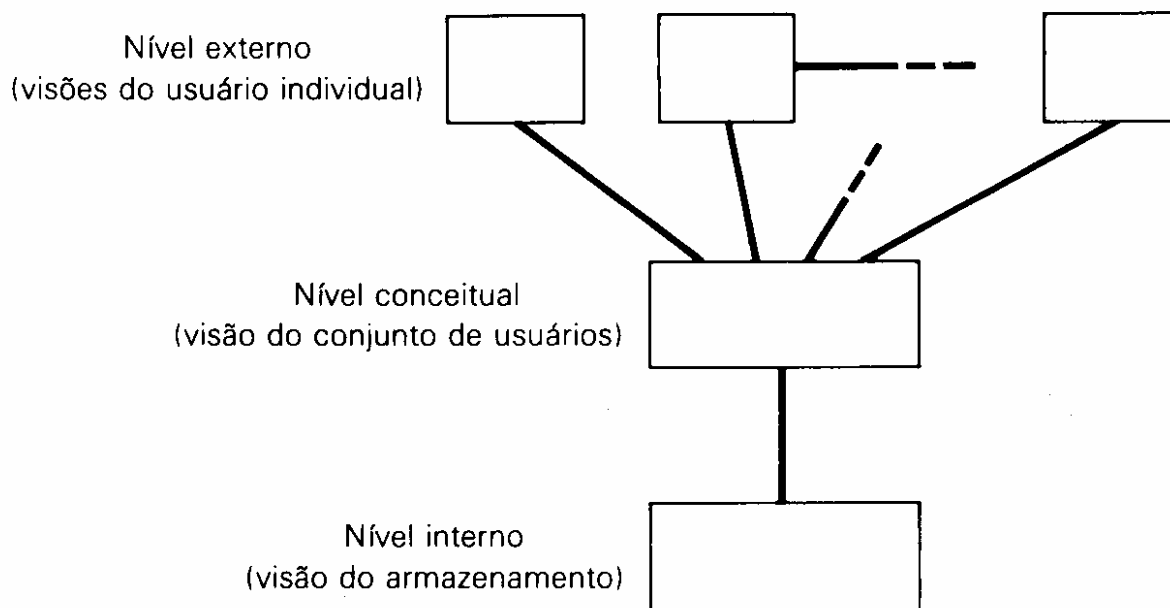
A arquitetura divide-se em três níveis gerais: interno, conceitual e externo (Figura 3). Em termos amplos:

1. o nível interno é o mais próximo ao armazenamento físico - i.e. relaciona-se à forma como são realmente armazenados os dados;
2. o nível externo é o mais próximo aos usuários - i.e., à forma como os dados são vistos pelos usuários individuais; e
3. o nível conceitual é o "nível de simulação", entre os dois outros. Se o nível externo diz respeito às visões do usuário individual, o nível conceitual pode ser considerado a visão do grupo de usuários.

Se o nível externo diz respeito às visões do usuário *individual*, o nível conceitual pode ser considerado a visão do grupo de usuários.

Em outras palavras, haverá muitas "visões externas" distintas, cada uma consistindo em uma representação mais ou menos abstrata de determinada parte do banco de dados e haverá precisamente uma "visão conceitual", que corresponde à representação abstrata do banco de dados em sua totalidade.' (Lembremo-nos que a maioria dos usuários não estará interessada em todo o

banco de dados, mas somente numa parte restrita do mesmo.) Da mesma forma, haverá exatamente uma "visão interna", representando todo o banco de dados como armazenado de fato.



**FIGURA 3 – Os três níveis da arquitetura**

Um exemplo tornará estas idéias mais claras. A figura 4 mostra a visão conceitual de um simples banco de dados sobre funcionários, a visão interna correspondente e as duas visões externas correspondentes, uma para um usuário de PL/I e outra para o usuário de COBOL. O exemplo, na certa, é totalmente hipotético - não pretende simular qualquer sistema real - e muitos detalhes irrelevantes foram deliberadamente omitidos.

Interpretamos a Figura 4 como segue:

<i>Externo (PL/I)</i>		<i>Externo (COBOL)</i>	
DCL	1 EMP,	01	EMPC.
	2 EMP # CHAR (6) ,	02	EMPNO PIC X (6)
	2 SAL FIXED BIN (31) ;	02	DEPTNO PIC X (4).
-----			
<i>Conceitual</i>			
EMPLOYEE			
	EMPLOYEE_NUMBER		CHARACTER (6)
	DEPARTAMENT_NUMBER		CHARACTER (4)
	SALARY		NUMERIC (5)
-----			
<i>Interno</i>			
STORED_EMP	LENGTH = 118		
PREFIX	TYPE = BITE(6), OFFSET = 0		
EMP #	TYPE = BYTE(6), OFFSET = 6, INDEX = EMPX		
DEPT #	TYPE = BYTE(4), OFFSET = 12		
PAY	TYPE = FULLWORD, OFFSET = 16		

**FIGURA 4 – Exemplo dos três níveis**

- O banco de dados contém, no nível conceitual, informações referentes ao tipo de entidade chamada EMPLOYEE. Cada EMPLOYEE tem um EMPLOYEE\_NUMBER (seis caracteres),

um DEPARTMENT NUMBER (quatro caracteres) e um SALARY (cinco dígitos decimais). Os funcionários estão representados, no nível interno, por um tipo de registro armazenado chamado STORED EMP, com dezoito bytes de comprimento. O STORED\_EMP contém quatro tipos de campos armazenados: um prefixo de seis bytes (contendo, provavelmente, informações de controle como sinalizadores ou ponteiros, e três campos de dados correspondendo às três propriedades dos funcionários, Os registros STORED\_EMP são, adicionalmente, indexados no campo CAMPO EMP por um índice chamado EMPX.

- O usuário de PL/I tem uma visão externa do banco de dados, na qual cada funcionário é representado por um registro PL/I, contendo dois campos (os números de departamento não são do interesse deste usuário, e por isto foram omitidos da visão). O tipo de registro é definido por uma declaração comum de estrutura PL/I, de acordo com as regras normais de PL/I.
- Do mesmo modo, o usuário de COBOL tem uma visão externa, na qual cada funcionário é representado por um registro COBOL, contendo, novamente, dois campos (desta vez foi omitido o de salário). O tipo de registro é definido por um registro comum COBOL, de acordo com as regras normais do COBOL.
- Observemos que objetos correspondentes podem ter nomes diferentes em cada nível. O número do funcionário, por exemplo, é chamado de EMP # na visão da PL/I, e de EMPNO na visão COBOL, como EMPLOYEE\_NUMBER na visão conceitual e como EMP # (novamente) na visão interna. O sistema certamente deve estar a par das correspondências. Deve ser informado, por exemplo, que o campo COBOL EMPNO deriva do objeto conceitual EMPLOYEE\_NUMBER que, por sua vez, é representado no nível interno pelo campo armazenado EMP # . Tais concordâncias, ou mapeamentos, não são mostradas na Figura 4.

Primeiro, o nível conceitual em tal sistema será definitivamente relacional, no sentido de que os objetos visíveis neste nível serão tabelas relacionais (os operadores também serão operadores relacionais, i.e., operadores que trabalham com tais tabelas). Segundo, uma determinada visão externa será igualmente relacional ou algo bem próximo; por exemplo, os registros PL/I e COBOL podem ser considerados, respectivamente, como as representações PL/I e COBOL de (uma linha dentro de) uma tabela relacional. Terceiro, o nível interno certamente não será sempre "relacional", desde que os objetos desse nível não serão exatamente tabelas relacionais (armazenadas) - ao contrário, serão a mesma espécie de objetos encontrados no nível interno de outros tipos de sistema (a saber, registros armazenados, ponteiros, índices,

acessos hash etc.). De fato, a teoria relacional como tal não tem nada a dizer sobre o nível interno (repetimos, sobre como o banco de dados aparece para o usuário).

Examinaremos agora mais detalhadamente os três níveis da arquitetura, iniciando com o nível externo.

## 6 - O NÍVEL EXTERNO

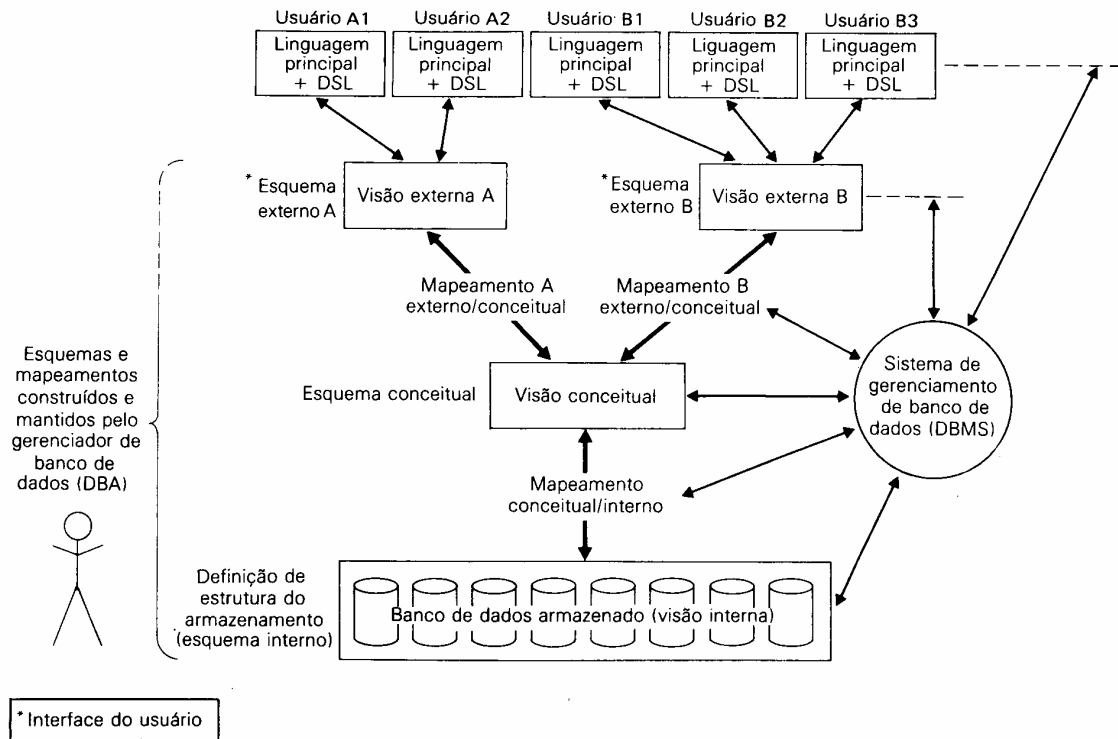
O nível externo é o nível do usuário individual. Um determinado usuário tanto pode ser um programador de aplicações como um usuário de terminal on-line - i.e., um usuário final - de qualquer grau de sofisticação. O DBA é um caso especial importante. (Ao contrário dos usuários comuns, o DBA terá de se interessar pelos níveis conceitual e interno também.

Cada usuário tem uma linguagem à sua disposição:

- Esta linguagem, para o programador de aplicação, pode ser uma linguagem convencional de programação, como COBOL ou PL/I, ou uma linguagem de programação apropriada, específica do sistema em questão (sistemas NOMAD, Rdv/VMS ou dBase II).

- Para o usuário final, pode ser uma linguagem de consulta ou uma linguagem de propósitos especiais, talvez baseada em formulários ou menu, modelada às necessidades do usuário e suportada por um programa de aplicação on-line.

Para nossos propósitos, o que importa é sabermos que todas essas linguagens incluirão uma sublinguagem de dados - ou seja, um subconjunto de toda a linguagem, voltado para os objetivos e operações do banco de dados. A sublinguagem de dados (DSL) é embutida na linguagem hospedeira correspondente, a qual proporciona os diversos recursos não-específicos de banco de dados, tais como variáveis locais (temporárias), operações computacionais, lógica if-then-else e assim por diante. Um determinado sistema pode suportar múltiplas linguagens hospedeiras e múltiplas sublinguagens de dados.



**FIGURA 5 – Sistema de Arquitetura detalhado**

Observação: Embora seja conveniente diferenciar, na arquitetura, a sublinguagem de dados e linguagem hospedeira, as duas, em relação ao usuário, podem ser indistinguíveis. Se assim for, ou se só puderem ser separadas com dificuldades, dizemos que são "unidades de maneira firme". Se as linguagens são fáceis e claramente separadas, dizemos que são "unidas de maneira indefinida". A maioria dos sistemas de hoje suporta apenas a união indefinida. Um sistema firmemente unido propiciará um conjunto de recursos mais uniforme para o usuário, mas obviamente requer maior esforço por parte dos projetistas do sistema (o que poderia explicar o que temos hoje). Entretanto, parece haver um movimento que, gradualmente, nos levará a dispor, nos próximos anos, de sistemas com união mais firme.

Em princípio, qualquer sublinguagem de dados é realmente uma combinação de pelo menos duas linguagens subordinadas: a linguagem de definição de dados (DDL), que possibilita a definição ou descrição dos objetos do banco de dados, e a linguagem de manipulação de dados (DML), que suporta a manipulação ou processamento desses objetos. Considerando-se o usuário de PL/I da figura 4, a sublinguagem de dados para aquele usuário compõe-se dos aspectos de PL/I utilizados para a comunicação com o DBMS. A parte DDL consiste nas construções declarativas do PL/I necessárias para declarar os objetos do banco de dados: a própria instrução DECLARE (DCL), certos tipos de dados PL/I, e possivelmente as extensões espe-

ciais para PL/I, para suportar novos objetos que não são tratados pelo PL/I existente. A parte DML compõe-se das instruções executáveis do PL/I que transferem as informações de e para o banco de dados - podendo, novamente, incluir novas instruções especiais. (Observação: Os PL/I atuais de fato não incluem aspectos específicos de banco de dados. As instruções "DML", por isto, são apenas "CHAMADAS" para o DBMS. Eis por que sistemas PL/I, como muitos sistemas atuais, só proporcionam uma união muito indefinida entre a sublinguagem de dados e a hospedeira.)

Voltando à arquitetura: Já mencionamos que o usuário individual, por via de regra, só vai interessar-se por determinada parte do banco de dados; além disso, a visão que o usuário terá daquela parcela é, em geral, algo abstrato, em comparação à forma como os dados são fisicamente armazenados. Em termos ANSI/SPARC, a visão de determinado usuário é uma visão externa. A visão externa é, portanto, o conteúdo do banco de dados como visto por determinado usuário (ou seja, para aquele usuário, a visão externa é o banco de dados). Um usuário do Departamento de Pessoal, por exemplo, pode ver o banco de dados como uma coleção de eventos de registros do departamento, e uma coleção de eventos de registros de empregados (Podendo estar totalmente desinformado das ocorrências de registros de fornecedores e peças vistas pelos usuários do Departamento de Compras). Portanto, uma visão externa consiste, em geral, em ocorrências múltiplas, de múltiplos tipos de registro externo<sup>3</sup>. Um registro externo não é necessariamente o mesmo que um registro armazenado. A sublinguagem de dados do usuário é definida em termos de registros externos; por exemplo, uma operação de "recuperação de registro" da DML irá recuperar um evento de registro externo, e não uma ocorrência de registro armazenado. Percebemos agora, conseqüentemente, que o termo "registro lógico" refere-se a um registro externo.

Cada visão externa é definida por meio de um esquema externo, que consiste, basicamente, em definições para cada um dos vários tipos de registro externo naquela visão externa. O esquema externo é descrito usando-se a parte DDL da sublinguagem de dados do usuário. (Denomina-se, assim, a DDL, às vezes, de DDL externa.) O tipo de registro externo funcionário, por exemplo, pode ser definido como um campo de seis caracteres, número do funcionário, mais um campo de cinco dígitos decimais, 'salário', e assim por diante. Além disso, deve haver uma definição do mapeamento entre o esquema externo e o esquema conceitual fundamental

## 9 - O NÍVEL CONCEITUAL

A visão conceitual é a representação de todo o conteúdo de informações do banco de dados, também (como a visão externa) um tanto abstrata quando comparada à forma como os

dados são fisicamente armazenados, que também pode ser bem diferente da maneira como os dados são vistos por qualquer usuário em particular. A grosso modo, podemos dizer que a visão conceitual é a visão dos dados "como realmente são", e não como os usuários são forçados a vê-los devido às restrições (por exemplo) da linguagem ou do hardware utilizados pelos mesmos.

A visão conceitual consiste em ocorrências múltiplas de tipos múltiplos de registros conceituais. A mesma pode consistir, por exemplo, em uma série de ocorrências de registros de departamentos, mais um conjunto de ocorrências de registros de funcionários, ou de fornecedores de peças ... De um lado, um registro conceitual não é necessariamente o mesmo do que um registro externo e, de outro, nem o mesmo que um registro armazenado.

A visão conceitual é definida pelo esquema conceitual que inclui definições de todos os diversos tipos de registros conceitual. O esquema conceitual é escrito através de outra linguagem de definição de dados, a DDL conceitual. Para que se possa alcançar a independência de dados, essas definições da DDL conceitual não podem conter quaisquer considerações sobre a estrutura de armazenamento ou a estratégia de acesso - devem ser apenas definições das informações. Assim, não pode haver referência ao esquema conceitual para as representações do campo armazenado, seqüência de registro armazenado, indexação, acesso hash, ponteiros ou quaisquer outros detalhes de armazenamento/acesso. Se o esquema conceitual for realmente independente de dados, então os esquemas externos, que são definidos em termos do esquema conceitual, também serão necessariamente independentes de dados.

A visão conceitual, então, é a visão do conteúdo total do banco de dados, e o esquema conceitual é uma definição desta visão. Seria errado, porém, dizer-se que o esquema conceitual não é nada mais do que um conjunto de definições parecidas com simples definições de registros como encontrados, por exemplo, num programa COBOL. As definições no esquema conceitual devem incluir uma grande quantidade de aspectos, como controles de segurança e de integridade. Algumas autoridades na matéria ainda vão mais além, e sugerem que o objetivo final do esquema conceitual é descrever toda a empresa - não somente os dados operacionais, mas também como são usados: como transcorre o fluxo em cada ponto da empresa, como é usado em cada ponto, como se aplicam a auditoria ou outros controles em cada ponto, e assim por diante. Enfatizamos, no entanto, que hoje em dia nenhum sistema realmente suporta um nível conceitual que se aproxime deste grau de abrangência; na maioria dos sistemas existentes, o "esquema conceitual" é realmente pouco mais que uma simples união de todos os esquemas externos e individuais, com a provável adição de alguns controles de segurança e integridade. Mas tudo indica que os sistemas, no futuro, serão eventualmente mais sofisticados quanto ao suporte do nível conceitual. Discutiremos este tópico com mais profundidade no final desse livro (víde Capítulo 25).

## 10 - O NÍVEL INTERNO

O terceiro nível da arquitetura é o nível interno. A visão interna é uma pequena representação de todo o banco de dados; consiste em ocorrências múltiplas de tipos múltiplos de registros internos. O "registro interno" é termo ANSI/SPARC para a estrutura que temos denominado de registro armazenado (termo que continuaremos a empregar). A visão interna é um tanto distante do nível físico, uma vez que não trabalha em termos de registros físicos (também chamados páginas ou blocos), nem de considerações de dispositivos específicos tais como cilindro ou comprimento de trilha. (A visão interna assume basicamente um espaço de endereçamento linear e infinito. Os detalhes de como este espaço de endereçamento é mapeado até o armazenamento físico são altamente específicos a cada sistema, e deliberadamente omitidos de arquitetura.)

A visão interna é descrita por meio do esquema interno, que não só define os vários tipos de registros armazenados como também especifica os índices que existem, como os campos armazenados são representados, a seqüência física dos registros armazenados, e assim por diante. O esquema interno é preparado através de uma outra linguagem de definição de dados - a DDL interna.

Observamos, à parte, que, em certas situações excepcionais, os programas de aplicação - em particular, as aplicações de natureza "utilitária" podem operar diretamente no nível interno, ao invés do nível externo. Não é necessário dizer que esta prática não é recomendável; representa um risco de segurança (posto que o controle de segurança não é observado) e um risco de integridade (uma vez que os controles de integridade também não são observados), e o programa, além disso, torna-se dependente de dados, em algumas ocasiões, porém esta poderá ser a única forma de se obter a função ou a performance necessária - assim como o usuário de uma linguagem de programação de alto nível pode, ocasionalmente, precisar utilizar a linguagem de montagem (assembler) para satisfazer certos objetivos funcionais ou de desempenho.

## 11- MAPEAMENTOS

Referindo-nos novamente à Figura 5, observa-se dois níveis de mapeamento na arquitetura, um entre os níveis externo e conceitual do sistema e um entre os níveis conceitual e interno. O mapeamento conceitual interno define a correspondência entre a visão conceitual e o banco de dados armazenado; especifica como os registros e campos conceituais são representados no nível interno. Se a estrutura do banco de dados armazenado for modificada - i.e., se for executada uma mudança na definição da estrutura armazenada -, o mapeamento concei-

tual/interno também deverá ser modificado de acordo, de forma que o esquema conceitual permaneça invariável. (O controle destas mudanças é da responsabilidade do DBA.) Em outras palavras, os efeitos dessas modificações devem ser isolados abaixo do nível conceitual, de maneira a preservar a independência de dados.

Um mapeamento externo/conceitual define a correspondência entre uma determinada visão externa e a visão conceitual. As diferenças que podem existir entre estes dois níveis são similares aquelas que podem existir entre a visão conceitual e o banco de dados armazenado. Como exemplo, os campos podem ter tipos de dados diferentes, as denominações de campo e registro podem ser modificadas, campos conceituais múltiplos podem ser combinados num único campo externo (virtual) etc. Pode haver qualquer número de visões externas ao mesmo tempo; qualquer quantidade de usuários pode compartilhar de uma determinada visão externa; diferentes visões externas podem sobrepor-se. Alguns sistemas possibilitam que a definição de uma visão externa seja expressa em termos de outra (de fato, via mapeamento externo/externo), ao invés de exigirem sempre uma definição explícita do mapeamento a nível conceitual - um aspecto muito útil, quando diversas visões externas se relacionam entre si.

## 12 - O ADMINISTRADOR DO BANCO DE DADOS

O administrador do banco de dados (DBA), já mencionado é a pessoa (ou grupo de pessoas) responsável pelo controle do sistema. As responsabilidades do DBA incluem o seguinte:

- Decidir o conteúdo de informações do banco de dados

Faz parte do trabalho do DBA decidir exatamente que informação manter no banco de dados - em outras palavras, deve identificar as entidades do interesse da empresa e a informação a registrar em relação a estas entidades. Uma vez feito isto, o DBA deve então definir o conteúdo do banco de dados, descrevendo o esquema conceitual (usando o DDL conceitual). A forma objeto (compilado) daquele esquema é utilizada pelo DBMS para responder às solicitações de acesso. A forma (não compilada) atua como documento de referência para os usuários do sistema.

- Decidir a estrutura de armazenamento e a estratégia de acesso

O DBA também deve decidir como os dados serão representados no banco de dados, e definir esta representação escrevendo a definição da estrutura de armazenamento (usando a DDL interna). Além disso, deve definir o mapeamento associado entre os níveis interno e conceitual. Na prática, tanto a DDL conceitual quanto a DDL interna - mais provavelmente a primeira - provavelmente incluirão os meios de definição desse mapeamento, mas as duas funções (a definição do esquema, a definição do mapeamento) devem ser claramente separadas. Tal como o esquema conceitual, o esquema interno e o mapeamento correspondente existirão não só na forma-fonte como na forma objeto.

- Servir de elo de ligação com usuários

É função do DBA servir de elo de ligação com os usuários, a fim de garantir a disponibilidade dos dados de que estes necessitam, e prepará-los ou auxiliá-los na preparação dos necessários esquemas externos, utilizando a DDL externa apropriada (como já mencionamos, um determinado sistema pode suportar diversas DDLs externas e distintas). E, ainda, também deve ser definido o mapeamento entre qualquer esquema externo e o esquema conceitual. Na prática, a DDL externa provavelmente incluirá os meios de especificação do mapeamento, mas o esquema e o mapeamento devem ser claramente separados. Cada esquema externo e o mapeamento correspondente existirão tanto na forma-fonte como na forma objeto.

- Definir os controles de segurança e integridade

Os controles de segurança e de integridade, como já mencionado, podem ser considerados parte do esquema conceitual. A DDL conceitual incluirá os recursos para a especificação de tais controles.

- Definir a estratégia de reserva e recuperação

A partir do momento em que a empresa começa efetivamente a basear-se em banco de dados, torna-se dependente do bom funcionamento deste sistema. Na eventualidade de danos à parte do banco de dados - causados, digamos, por erro humano, ou por falha no hardware, ou no sistema operacional de suporte -, é de suma importância fazer retornar os dados envolvidos com um mínimo de demora e com as menores conseqüências ao restante do sistema. Por exemplo, os dados que não sofreram danos não deverão ser afetados. O DBA deve definir e implementar uma estratégia de recuperação apropriada envolvendo, por exemplo, o descar-

regamento periódico do banco de dados na memória auxiliar de armazenamento e procedimentos para recarregá-lo, quando necessário.

- Monitorar o desempenho e atender as necessidades de modificações.

O DBA deve organizar o sistema de tal maneira que obtenha "o melhor desempenho para a empresa"; e efetuar os ajustes adequados quanto às necessidades de modificações. Como já mencionamos, quaisquer mudanças nos detalhes de armazenamento e acesso devem ser acompanhadas de mudanças correspondentes na definição do mapeamento, a partir do nível conceitual, de forma que o esquema conceitual possa permanecer constante. O DBA, evidentemente, irá precisar de diversos programas utilitários como auxílio às tarefas precedentes. Estes são uma parte essencial de um sistema de banco de dados prático, embora não sejam mostrados na Figura 5 da arquitetura. Listamos abaixo alguns exemplos dos programas utilitários necessários.

- Rotinas de carga (para criar uma versão inicial do banco de dados a partir de um ou mais arquivos).
- Rotinas de despejo na memória e recuperação (despejar o banco de dados, auxiliar de armazenamento de dados, e recarregar o banco de dados a partir dessa cópia de segurança). Observação: As rotinas de carga mencionadas acima consistirão, na prática, no aspecto de "recuperação" das rotinas de despejo/recuperação na memória. Rotinas de reorganização (para rearrumar os dados no banco de dados, em vista de diversas razões de desempenho - por exemplo, agrupar os dados de certa maneira ou regenerar espaço ocupado por dados que se tornaram obsoletos).
- Rotinas estatísticas (para computar diversos desempenhos estatísticos, tamanhos de arquivos e distribuição de valores de dados). Rotinas analíticas (para analisar as estatísticas mencionadas). Uma das ferramentas mais importantes do DBA - de muitas maneiras e, de fato, o coração de todo o sistema, embora não mostrado na Figura 5 - é o dicionário de dados (conhecido também como catálogo do sistema). O dicionário de dados pode ser considerado um banco de dados (mas um banco de dados de sistema, e não propriamente um banco de dados de usuário). O conteúdo do dicionário pode ser considerado "dados sobre dados" (às vezes denominado "metadados") - ou seja, descrições de outros objetos no sistema, ao invés de simples "dados em bruto". Os diversos esquemas e mapeamentos (externo, conceitual etc.), especialmente, serão fisicamente armazenados, ambos na forma-fonte e na forma objeto, no dicionário. Um di-

cionário abrangente também incluirá referências cruzadas das informações, mostrando, por exemplo, que programas utilizam tal parte do banco de dados, que departamentos necessitam de tais relatórios, que terminais estão conectados ao sistema, e assim por diante. O dicionário também pode (e deveria) estar integrado ao banco de dados que descreve, incluindo, portanto, a sua própria descrição. Deveria ser possível consultar-se o dicionário, tal como qualquer outro banco de dados, de maneira que fosse possível indicar os programas e/ou usuários que seriam afetados por eventual mudança no sistema.

### 13- O SISTEMA DE GERENCIAMENTO DO BANCO DE DADOS.

O sistema de gerenciamento do banco de dados (DBMS) é o software que manipula todos os acessos ao banco de dados. De forma conceitual, acontece o seguinte:

1. O usuário emite uma solicitação de acesso, usando uma sublinguagem específica de dados (por exemplo, SQL).
2. O DBMS intercepta a solicitação e analisa-a.
3. O DBMS, por sua vez, inspeciona os esquemas externos para aquele usuário, o mapeamento externo/conceitual correspondente, o esquema conceitual, o mapeamento conceitual/interno e a definição da estrutura de armazenamento.
4. O DBMS executa as operações necessárias no banco de dados armazenado.

Consideramos, por exemplo, o que envolve a recuperação de uma ocorrência específica de registro externo. Em geral, diversas ocorrências de registro conceitual vão solicitar campos. Cada ocorrência de registro conceitual, por sua vez, pode solicitar os campos de diversas ocorrências de registro armazenado. O DBMS, finalmente, precisa, em primeiro lugar, recuperar todas as ocorrências de registro armazenado solicitadas, depois construir as ocorrências de registro conceitual solicitadas, e então construir a ocorrência de registro externo solicitada. Em cada estágio, são necessários tipos de dados ou outras conversões.

Consideremos, entretanto, que a descrição acima está muito simplificada. A mesma sugere, especialmente, que todo o processo é interpretativo, o que implica, em geral, desempenho fraco (improdutividade do tempo de execução). As solicitações de acesso, na prática, talvez possam ser compiladas antes do tempo de execução.

Podemos caracterizar, de outra maneira, a função do DBMS, ou seja, o mesmo proporciona a interface de usuário ao sistema de banco de dados. A interface do usuário pode ser definida como um limite no sistema abaixo do qual tudo é invisível ao usuário. Por definição, a interface do usuário está no nível externo, há certas situações em que a visão externa difere de maneira muito significativa da visão conceitual fundamental (a parte relevante de).

#### 14- COMUNICAÇÕES DE DADOS

Concluimos este capítulo com uma breve menção à comunicação de dados. As solicitações ao banco de dados a partir de um usuário final são normalmente transmitidas (a partir do terminal do usuário -- que pode estar fisicamente longe do sistema - para alguma aplicação on-line, embutida ou outra, e dali para o DBMS) na forma de mensagens de comunicação. As respostas ao usuário (do DBMS e da aplicação on-line de volta ao terminal do usuário) também são transmitidas sob a forma de mensagens. Todas essas transmissões de mensagens são efetuadas sob a direção de um outro sistema de software, o gerenciador de comunicação dos dados (gerenciador DC).

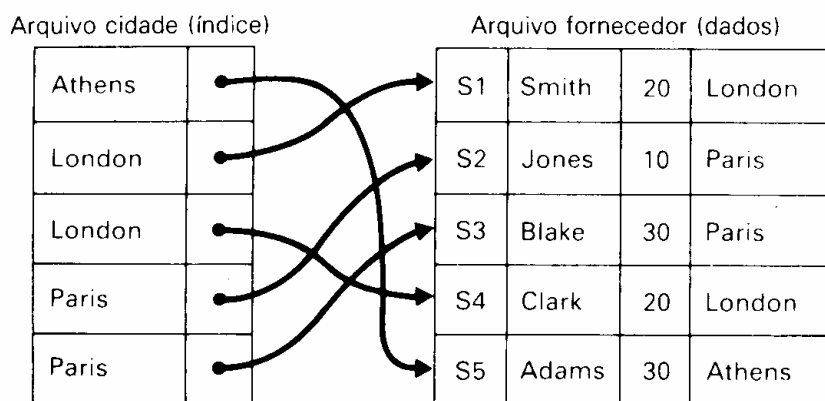
O gerenciador DC não é um componente do DBMS, mas sim um sistema autônomo em si. Entretanto, posto que gerenciador DC e o DBMS devem trabalhar harmoniosamente, costumamos considerá-los parceiros de mesmo nível no exercício denominado banco de dados/sistema de comunicação de dados (sistema DB/DC); o DBMS procura o banco de dados e o gerenciador DC manipula todas as mensagens para e

As diversas técnicas não devem ser consideradas passíveis de se excluírem mutuamente. Por exemplo, é perfeitamente possível ter um arquivo armazenado (digamos) tanto com acesso indexado como com acesso hash ao arquivo baseado no mesmo campo armazenado, ou com acesso hash em um campo e um acesso de cadeia de ponteiro em outro.

#### 15 - INDEXAÇÃO

Consideremos a tabela de fornecedores na Figura 6, mais uma vez. Suponhamos a consulta: "Achar todos os fornecedores da cidade C" (onde C seja um parâmetro). Esta consulta é importante - i.e., será executada com frequência e, por isso, deve ser bem resolvida. Assim, o DBA talvez escolha a representação armazenada mostrada na Fig. 3.10, onde existem dois arquivos armazenados, um arquivo de fornecedores, e um arquivo de cidades (provavelmente em diferentes conjuntos de páginas); o arquivo de cidades, armazenado na seqüência de cidades (porque CITY é uma chave primária, inclui ponteiros (RIDs) ao arquivo de fornecedores. Para que o DBMS possa achar todos os fornecedores de Londres (digamos), existem duas estratégias possíveis:

1. Buscar em todo o arquivo de fornecedores, procurando todos os registros cujo valor de cidade seja igual a Londres.



**FIGURA 6 – Indexação de arquivos de fornecedores em CITY**

2. Buscar as entradas de Londres no arquivo de cidades e, para cada uma, acompanhar o ponteiro ao registro correspondente no arquivo de fornecedores. Se a proporção de fornecedores de Londres em relação aos outros for pequena, provavelmente a segunda estratégia seria mais eficaz do que a primeira, porque 1) o DBMS está a par do seqüenciamento físico do arquivo de cidade (pode parar a busca naquele arquivo, assim que o mesmo achar uma cidade que suceda Londres na ordem alfabética), e 2) mesmo que tenha de empreender uma busca em todo o arquivo de cidades, a mesma ainda necessitaria de menos entradas/saídas, pois o arquivo de cidades é fisicamente menor do que o de fornecedores (pois os registros são menores).

O arquivo de cidades neste exemplo é tido como um índice para o arquivo de fornecedores; o arquivo de fornecedores, da mesma forma, é tido como indexado pelo arquivo de cidades. Um índice é uma espécie especial de arquivo armazenado. Para ser mais específico, é um arquivo no qual cada entrada (i.e., registro) compõe-se precisamente de dois valores, um valor de dados e um ponteiro (RID);

o valor de dados é um valor para determinado campo do arquivo indexado, e o ponteiro identifica um registro daquele arquivo que tenha o valor daquele campo.

A partir deste ponto, passaremos a nos referir ao arquivo de cidades da Figura 6 mais explicitamente como "o índice CITY". Um ponto da terminologia: Um índice de um campo de chave primária - por exemplo, um índice no campo S # do arquivo de fornecedores - é chamado de índice primário. Um índice de qualquer outro campo - por exemplo, o índice CITY do exemplo - é denominado índice secundário. Como São Usados os Índices.

A vantagem fundamental do índice é que acelera a recuperação. Entretanto há também uma desvantagem - reduz a velocidade das atualizações. (Como em muitas outras situações, há uma compensação.) Por exemplo, cada vez que se adiciona um novo registro armazenado ao arquivo indexado, uma nova entrada terá de ser acrescentada ao índice. Consideremos, como

exemplo mais específico, o que o DBMS deve fazer no índice CITY da Fig. 6 para mudar o fornecedor S2 de Paris para Londres. A questão que deve ser respondida quando um campo é considerado como candidato à indexação é: O que seria mais importante, a recuperação eficiente baseada no valor do campo em questão ou a perda em atualização, em função da recuperação eficiente?

No restante desta seção, passaremos a concentrar-nos especificamente nas operações de recuperação.

Os índices podem ser utilizados, essencialmente, de duas maneiras diferentes. Primeiro, podem ser utilizados para o acesso seqüencial ao arquivo indexado - onde "seqüencial" significa "na seqüência definida pelos valores do campo indexado". Por exemplo, o índice CITY no exemplo acima permite que os registros no arquivo de fornecedores sejam processados na seqüência de cidades. Segundo, os índices também podem ser utilizados para acesso direto aos registros individuais no arquivo indexado baseado num determinado valor do campo indexado. A consulta "achar os fornecedores em Londres", discutida ilustra o segundo caso.

Os dois meios básicos de utilizar um índice podem ser delineados, simplesmente como:

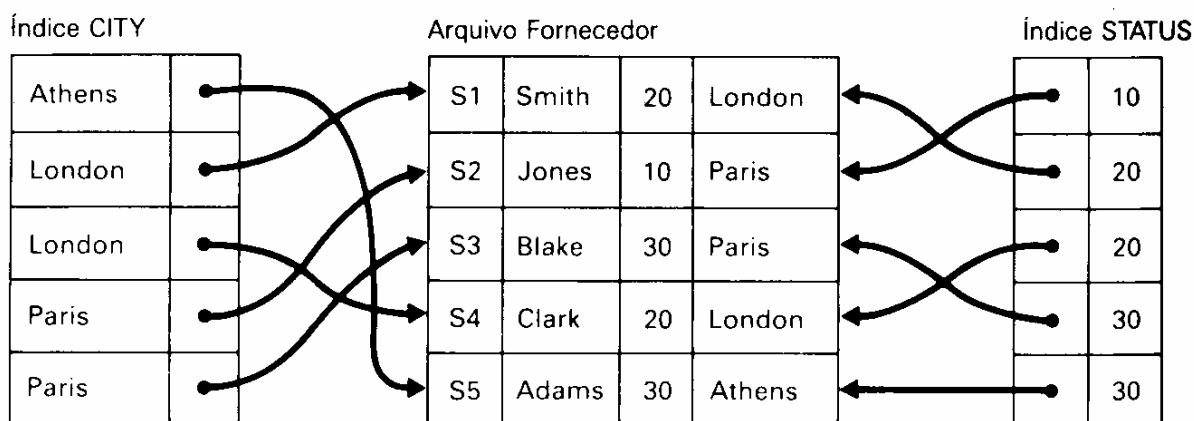
1. Seqüencial: O índice também pode ser útil nas consultas de limites - por exemplo, "Achar os fornecedores cuja cidade se encontre em determinado limite do alfabeto" (isto é, inicie-se com a letra no limite L-R).

2. Direto: O índice também pode ser útil nas consultas de lista - por exemplo, "Achar os fornecedores cuja cidade se encontre em certa lista específica" (isto é, a lista da cidade de Londres, Paris e Nova York).

Há, ainda, certas consultas - basicamente, testes de existência - que podem ser respondidas apenas pelo índice, sem qualquer acesso ao arquivo indexado. Considere como exemplo a consulta: "Há fornecedores em Atenas?".

A resposta é claramente "sim" se, e apenas se, existir uma entrada para Atenas no índice CITY.

Um determinado arquivo armazenado pode ter qualquer quantidade de índices. Por exemplo, o arquivo armazenado de fornecedores pode ter um índice CITY e um índice STATUS (Figura 7).



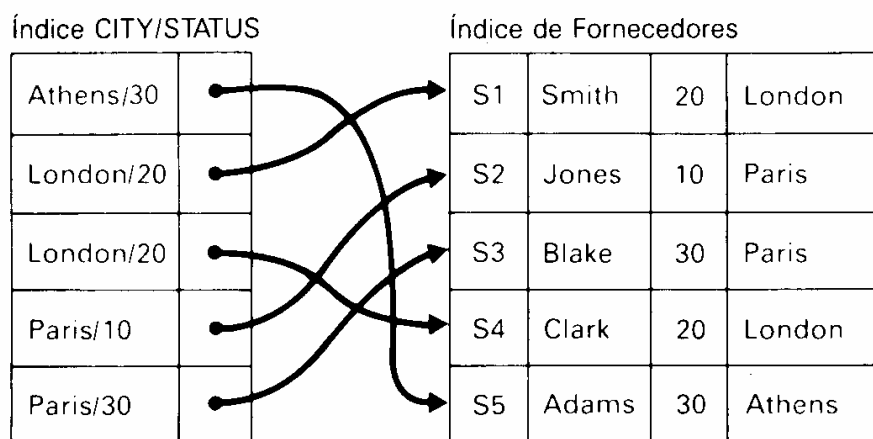
Estes índices poderiam então ser utilizados para proporcionarem acesso eficiente aos registros de fornecedores na base de determinados valores para cada ou ambos os índices CITY e STATUS. Como um exemplo do caso "ambos", consideremos a consulta: "Achar os fornecedores em Paris com status 30". O índice CITY fornece os RIDs - r2 e r3, digamos - para os fornecedores de Paris; da mesma maneira, o índice STATUS fornece os RIDs - r3 e r5, digamos - para os fornecedores de status 30. Deduzimos desses dois conjuntos de RIDs que o único fornecedor que satisfaz a consulta original é o fornecedor com RID igual a r3 (ou seja, o fornecedor S3). Somente, então, o DBMS acessa o arquivo de fornecedores em si, a fim de recuperar o registro desejado.

### **FIGURA 7 – Indexação de arquivo de fornecedores tanto em CITY como em STATUS**

Mais terminologia: Denominam-se os índices, eventualmente, de *listas invertidas*, pela seguinte razão: Primeiro um arquivo “normal” – o arquivo de fornecedores das figuras 6 e 7 é considerado como o típico “arquivo normal” – lista, para cada registro, os valores de campo naquele registro. Por outro lado, o índice lista, para cada valor de campo indexado, os registros que contêm aquele valor. Mais um termo: Um arquivo com um índice em cada campo é denominado, eventualmente, de completamente invertido.

#### **Indexação de Combinações de Campos**

Também é possível construir um índice baseado nos valores de dois ou mais campos combinados. Por exemplo, a Fig. 7 mostra um índice para o arquivo de fornecedores, combinando os campos CITY e STATUS (nesta ordem). O DBMS, com tal índice, pode responder à consulta discutida acima - "Achar os fornecedores de Paris com status 30" - numa simples exploração de índice único. Se o índice combinado for recolocado por dois índices separados, a consulta, então, envolverá duas explorações separadas de índice (como descrito). Ademais, nesse caso talvez seja difícil decidir quais das duas explorações deve ser efetuada em primeiro lugar, uma vez que as duas seqüências possíveis podem ter características muito diferentes em desempenho, cuja escolha seria bastante significativa.



**FIGURA 8 – Indexação do arquivo de fornecedores na combinação de campo CITY/STATUS**

Observemos que o índice combinado CITY/STATUS também pode servir como índice para o campo de CITY apenas, desde que todas as entradas para uma determinada cidade sejam consecutivas dentro do índice combinado. (Entretanto, um outro índice separado deverá ser feito, caso seja necessária a indexação de STATUS.) Em geral, um índice para a combinação de campos F1, F2, F3 ..., Fn (nesta ordem) também servirá como índice de F1 apenas, como índice das combinações F1F2 (ou F2F1), como índice da combinação F1F2F3 (em qualquer ordem), e assim por diante. A quantidade total de índices necessários para a indexação, deste modo, não é tão grande como pode parecer à primeira vista.

### 15.1 – INDEXAÇÃO DENSA VERSUS INDEXAÇÃO NÃO DENSA

Como mencionado em diversas ocasiões, o propósito fundamental do índice é acelerar a recuperação dos dados - mais especificamente, reduzir o número de entradas/saídas em disco necessárias para a recuperação de determinado registro armazenado. Atinge-se, basicamente, este propósito por meio de ponteiros; e a partir de agora assumimos que todos os ponteiros são ponteiros de registro (ou seja, RID's). De fato seria suficiente para este propósito que estes fossem simples ponteiros de páginas (isto é, números de página). Realmente, o sistema teria um trabalho adicional para achar um registro numa determinada página, pois precisaria explorar através das páginas do armazenamento principal, embora a quantidade de entradas/saídas permanecessem as mesmas.

Levemos a idéia adiante. Lembremo-nos de que qualquer arquivo armazenado tem uma única seqüência "física", representada pela combinação de 1) seqüência de registros armazenados em cada página e 2) seqüência de páginas dentro do conjunto de páginas. Supon-

hamos que o arquivo de fornecedores seja armazenado tal como a seqüência física corresponde à seqüência lógica, como definido pelos valores de determinado campo, digamos o campo de número de fornecedores; em outras palavras, o arquivo de fornecedores (Figura 9) é agrupado naquele campo. Suponhamos, também, que seja necessário um índice naquele campo. Não há necessidade de que aquele índice inclua uma entrada para cada registro armazenado no arquivo indexado (isto é, o arquivo de fornecedores, no exemplo). Tudo o que é necessário é uma entrada para cada página, determinando o número mais alto de fornecedor na página e o número correspondente da página.

Consideremos, como exemplo, o que é necessário para recuperar o fornecedor S3 utilizando este índice. O sistema primeiramente precisa explorar o índice, procurando a primeira entrada com o número de fornecedores superior ou igual a S3. O sistema acha a entrada indexada para o fornecedor S4, que aponta a página p (digamos), recupera-a e explora-a no armazenamento principal, à cata do desejado registro armazenado (que neste exemplo será encontrado rapidamente).

Um índice, como o da Fig. 9, é denominado não-denso, pois não contém uma entrada para todo registro armazenado no arquivo indexado. (Todos os índices até então discutidos, pelo contrário, eram índices densos.) Uma das vantagens do índice não-denso é que ocupará menos armazenamento que o índice denso correspondente, pela óbvia razão de que contém menos entradas. Em consequência, a exploração provavelmente será mais rápida. Como desvantagem, não possibilitará o desempenho de testes de existência com base no índice apenas.

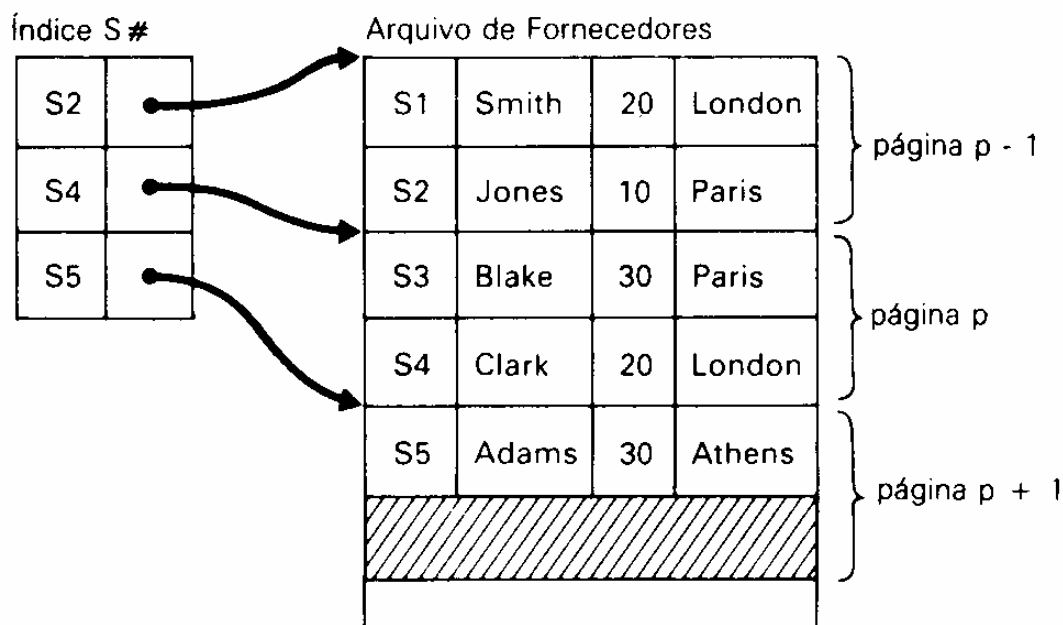


Figura 9 – Exemplo de um índice não-denso

Observemos que um determinado arquivo armazenado não pode ter mais do que um índice não-denso, pois o mesmo baseia-se na (única) seqüência física do arquivo em questão. Todos os outros índices, necessariamente, devem ser densos.

## 15.2 - Árvores B

Um tipo de índice particularmente comum e importante é a árvore B. Embora seja verdade (como já observado) de que não existe uma única estrutura de armazenamento ótima para todas as aplicações, não há dúvidas de que, se é necessário escolher uma única estrutura, então a árvore B, de uma variedade ou outra, será escolhida. As árvores B parecem ter o melhor desempenho. Assim, a maioria dos sistemas relacionais suportam as árvores B como a principal forma de estrutura de armazenamento, e outros não suportam absolutamente nada, a não ser estas. Antes que possamos explicar o que é uma árvore B, precisamos discutir uma noção preliminar, a saber, o índice multinível (ou estruturado em forma de árvore).

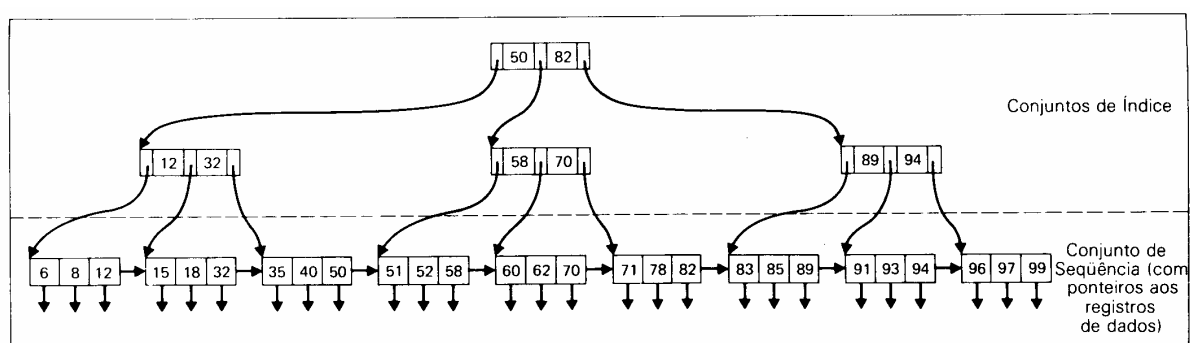
Providenciamos um índice em primeiro lugar para remover a necessidade de exploração física seqüencial do arquivo indexado. Entretanto a exploração física seqüencial ainda é necessária no índice. Se o arquivo indexado for muito grande, então o índice também terá um tamanho equivalente, e a exploração seqüencial do índice poderá tomar muito tempo. A solução para este problema é a mesma: ou seja, tratamos o índice simplesmente como um arquivo armazenado/regular e construímos um índice para o mesmo (um índice para o índice). Esta idéia pode ser colocada em prática em tantos níveis quanto necessário (em geral, três níveis na prática); um arquivo teria de ser muito grande para necessitar de mais do que três níveis de indexação). Cada nível de índice atua como um índice não-denso para o nível inferior (deve ser o não-denso, certamente, pois, de outra maneira, não se realizaria nada - o nível  $n$  conteria o mesmo número de entradas como o nível  $n + 1$ , e o tempo de exploração permaneceria o mesmo).

Agora podemos discutir a árvore B. A árvore B é um tipo especial de índice estruturado na forma de árvore, como descrito num texto de Bayer e McCreight, em 1972 [3.10]. Desde então, diversas variações foram propostas em relação à idéia básica, por Bayer e outros pesquisadores; como mencionado, as árvores B, de um ou outro modo, são agora provavelmente a estrutura de armazenamento mais utilizada nos sistemas modernos de banco de dados (relacionais ou não). Descrevemos a variação realizada por Knuth [3.1]. Mencionamos que a estrutura de índice do "Método de Acesso Virtual ao Armazenamento" VSAM da IBM [3.12] é muito similar à estrutura de Knuth. Contudo a versão VSAM inclui características próprias, como a utilização de técnicas (vide Seção 3.7). Na realidade, o precursor da estrutura VSAM já fora descrito por Chang em 1969 [3.13].

Na variação Knuth, o índice compõe-se de duas partes, o conjunto de seqüências e o conjunto de índices (terminologia VSAM).

1. O conjunto de seqüências consiste em um índice de nível único aos dados reais; em geral, é denso, mas pode ser não-denso, se o arquivo indexado for agrupado ao campo indexado. As entradas no índice são (certamente) agrupadas em páginas, e as páginas são (certamente) encadeadas, de forma que a ordem lógica representada pelo índice seja obtida através das entradas em ordem física na primeira página da cadeia, seguida das entradas na ordem física na segunda página da cadeia, e assim por diante. Desta maneira, o conjunto de seqüência proporciona um rápido acesso seqüencial aos dados indexados.

2. O conjunto de índices, por sua vez, proporciona um rápido acesso direto ao conjunto de seqüências (e, conseqüentemente, aos dados). O conjunto de índice é, na realidade, um índice estruturado em forma de árvore B para o conjunto de seqüências; o conjunto de índices representa a árvore B. A combinação do conjunto de índices e do conjunto de seqüências é denominado eventualmente, de "Árvore B-plus" (árvore B+). O nível superior do conjunto de índices consiste em um único nó (isto é, uma única página, mas contendo entradas múltiplas de



índice, como todos os outros nós). O nó superior é denominado raiz. A Fig. 10 mostra um exemplo simples. A Fig. 10 se exemplifica da seguinte forma: Os valores 6, 8, 12 ..., 97, 99 são o campo indexado, digamos, F. Consideremos o nó superior, que consiste em dois valores F (50 e 82) e três ponteiros (números de página). Os registros de dados com F menor ou igual a 50 podem ser achados (eventualmente) seguindo-se o ponteiro esquerdo a partir deste nó; da mesma forma, os registros com F, superiores a 50 e menores ou iguais a 82 podem ser achados seguindo-se o ponteiro do meio; os registros com F superiores a 82 são achados seguindo-se o ponteiro da direita. Os outros nós do conjunto de índices são interpretados de forma análoga; observemos que (por exemplo), ao seguirmos o ponteiro direito a partir do primeiro nó ao segundo nível, chegamos a todos os registros com F superior a 32 e também inferior ou igual a 50 (em virtude do fato de já termos seguido o ponteiro esquerdo a partir do nó superior).

## Figura 10 – Parte de uma árvore B simples

A árvore B (isto é, o conjunto de índices) da Fig. 10 é um tanto fora da realidade, pelas seguintes razões:

1. Primeiro, nem todos os nós de uma árvore B contêm o mesmo número de dados;
2. Segundo, os mesmos normalmente contêm um certo espaço livre. Em geral, uma árvore B da ordem  $n$  tem ao menos  $n$  mas não mais do que  $2n$  de valores de dados em qualquer nó (e se o mesmo tiver valores  $k$ , também tem  $k + 1$  ponteiro). Nenhum valor de dados aparece na árvore mais do que uma vez. Fornecemos o algoritmo de exploração para o valor especial  $V$  na estrutura;

O algoritmo para a árvore B da ordem  $n$  é uma simples generalização.

```
set N to the root node ;
repeat until N is a sequence-set node ;
Let X, Y be the data values in node N |* X < Y *| ;
if V <= X then set N to the left lower node of N ; if X < V <= Y then set N to the middle lower
node of N ;
if V > Y then set N to the right lower node of N ; end repeat ;
if V occurs in node N then exit /* found */ ;
if V does not occur in node N then exit /* not found */ ;
```

As estruturas de árvore em geral têm um problema, ou seja, tais inserções e eliminações podem causar o desbalanceamento da árvore. Uma árvore é desbalanceada quando todas as folhas de nós não se encontram no mesmo nível - isto é, se folhas diferentes de nódulos se encontram em diferentes distâncias do nó da raiz. Como a pesquisa da árvore envolve um acesso em disco para cada nó, a pesquisa em árvore desbalanceada pode ter uma duração imprevisível. A grande vantagem as árvores B é que o algoritmo de inserção/eliminação da mesma garante que a árvore estará sempre balanceada. (Diz-se que o "B" na designação "árvore B" significa balanceada por esta razão.) Vamos considerar brevemente a inserção de um novo valor, digamos  $V$ , na árvore de ordem  $n$ . O algoritmo como descrito suprime apenas o conjunto de índices, posto que, como explicado anteriormente, o conjunto de índices é a própria árvore B; para trabalhar com o conjunto de seqüências também é necessária uma extensão simples.

- Primeiramente, o algoritmo de pesquisa foi executado para localizar não o nó do conjunto de seqüências, mas sim aquele nó (digamos N) no nível mais inferior do conjunto de índices, ao qual V pertence logicamente. Se N contiver espaço livre, V será inserido em N, e o processo se encerra.

- Caso contrário, o nó N (que deverá conter valores  $2n$ ) será fracionado em dois nós, N1 e N2. S constitui o conjunto ordenado composto de valores originais  $2n$ , mais o novo valor V, em seqüência lógica.

Os valores  $n$  mais baixos do conjunto S serão colocados no nó esquerdo N1, os valores  $n$  mais elevados daquele conjunto serão colocados no nó direito N2, e o valor médio, digamos W, será deslocado para o nó-fonte de N, digamos P, a fim de servir como valor separador dos nós N1 e N2. As futuras buscas de valor V', ao alcançarem o nó P, serão direcionadas ao nó N1, se  $V' \leq W$ , e para o nó N2, se  $W < V'$ .

- Tenta-se, agora, inserir W em P, e o processo se repete.

No pior dos casos, ocorrerá um fracionamento até o topo da árvore; será citado um novo nó raiz (fonte da antiga raiz que agora estará fracionada em duas; e a árvore crescerá mais um nível de altura (mas, mesmo assim, continuará balanceada).

O algoritmo de eliminação é essencialmente o inverso do algoritmo de inserção acima descrito. A modificação de um valor é tratada pela eliminação do antigo valor e pela inserção do novo valor.

### 15.3- ACESSO HASH

O acesso hash (também denominado endereçamento-hash) é a técnica que proporciona um rápido acesso direto ao registro armazenado, baseado num determinado valor de um certo campo. O campo em questão é, em geral, mas não necessariamente, a chave primária. Esta técnica funciona como segue:

- Cada registro armazenado é colocado no banco de dados em uma localização, cujo endereço (RID, ou talvez apenas um número de página) é computado como função (a função hash) de algum campo daquele registro (o campo hash). O endereço computado é denominado endereço hash.

- Para armazenar o registro, inicialmente o DBMS computa o endereço hash para o novo registro e instrui o gerenciador de arquivo no sentido de colocar o registro naquela posição.

- Para recuperar o registro posteriormente, dado o valor do campo hash, o DBMS desempenha a mesma computação anterior, e instrui o gerenciador de arquivo no sentido de buscar o registro na posição computada.

Suponhamos, a título de ilustração, que 1) os valores dos números de fornecedores sejam S100, S200, S300, S400, S500 (ao invés de S1, S2, S3, S4 e S5) e, 2) cada registro de fornecedor armazenado necessite de uma página inteira, e consideremos a função hash:

endereçamento hash (i.e. número de página) = restante da divisão da parte numérica do valor S # por 13

- exemplo simples de um tipo muito comum da função hash, denominado "divisão/restante". (Por motivos que fogem ao escopo deste manual, escolhe-se um número primo como divisor de uma divisão/restante hash, usualmente, como em nosso exemplo). Os números de páginas para os cinco fornecedores são, então, 9, 5, 1, 10, 6, respectivamente, dando-nos a representação na figura 11.

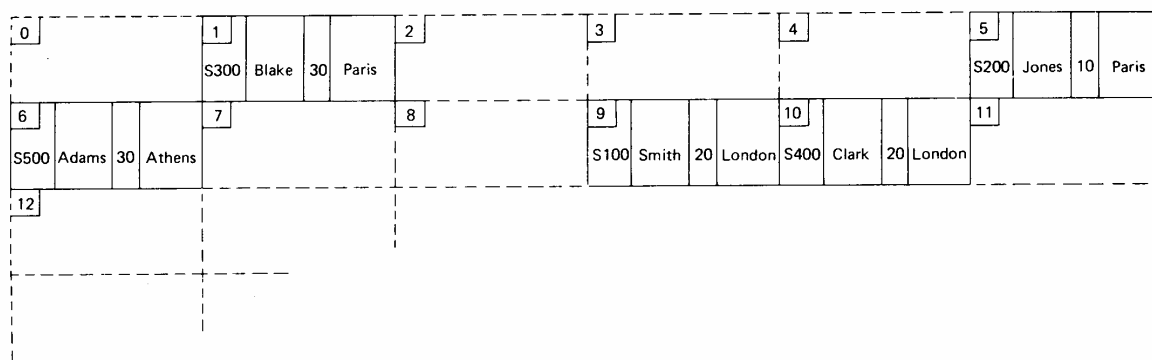
Deve ter ficado claro, pela descrição anterior, que o acesso hash difere da indexação, uma vez que um determinado arquivo armazenado pode ter qualquer quantidade de índices, mas só pode ter uma estrutura hash.

Dizendo de outra forma: Um arquivo pode ter qualquer quantidade de campos indexados, mas somente um campo hash.

Além de mostrar como trabalha o acesso hash, o exemplo também demonstra porque a função hash é necessária. Teoricamente seria possível usar uma função hash "identidade". Isto é, usar o valor (numérico) da chave primária para qualquer registro armazenado, diretamente como o endereço hash. Contudo tal técnica normalmente será inadequada na prática, porque a margem de valores de chave primária possíveis será, em geral, mais ampla do que a margem de endereços disponíveis. Suponhamos, por exemplo, que os números de fornecedores tenham, de fato, três dígitos a mais, como no exemplo acima. Haveria então 1000 possibilidades de números distintos de fornecedores, enquanto, de fato, existiriam apenas cerca de 10 fornecedores. Assim, para evitar uma perda considerável de espaço de armazenamento, o ideal seria descobrir uma função hash que reduza qualquer valor da faixa de 000-999 para a faixa 0-9 (digamos). Para permitir um pouco de espaço para o crescimento futuro, é normal que se aumente a margem em cerca de 20 por cento; foi por isto que escolhemos uma função que gera valores na faixa de 0-12, e não 0-9, como no exemplo acima.

O exemplo também ilustra uma das desvantagens do acesso hash: A "seqüência física" de registros no arquivo armazenado não será certamente a seqüência da chave primária, nem

qualquer outra que tenha qualquer interpretação lógica sensível. (Ademais, pode haver espaços vazios de tamanhos arbitrários entre os registros consecutivos.) De fato, em geral a seqüência física de um arquivo armazenado com estrutura hash é (não invariavelmente) tida como não-representativa de seqüência lógica específica, assim, via de regra, arquivo de acesso hash não tem nenhum agrupamento intra-arquivo - o que é de se lamentar, já que o agrupamento físico, como mencionado neste capítulo, quase sempre seria do maior interesse.



**Figura 11 – Exemplo de uma estrutura Hash**

Uma outra desvantagem do acesso hash é a possibilidade de colisões - isto é, de achar dois registros distintos que indiquem o mesmo endereço. Suponhamos, por exemplo, que o arquivo de fornecedores (com os fornecedores S100, S200 etc.) também incluía um fornecedor com número de fornecedor S1400. Dada a função hash "dividir por 13" discutida acima, aquele fornecedor colidiria (no endereço hash 9) com o fornecedor S 100.

A função hash, como o exemplo demonstra, é inadequada; ela deve ser aprofundada no sentido de lidar com o problema de colisão.

Em nosso exemplo original, uma possibilidade seria tratar o restante da divisão por 13, não como um endereço hash em si, mais como ponto de partida para uma busca seqüencial. Para inserir o fornecedor S1400 (supondo que já existam os fornecedores S100-S500), vamos à página 9 e buscamos à frente, a partir desta posição, a primeira página livre.

O novo fornecedor será armazenado na página 11. Para depois recuperar este fornecedor, usaremos um processo similar. Este método de busca linear será adequado se (como em geral ocorre na prática) os registros múltiplos estiverem armazenados em todas as páginas. Suponhamos que cada página possa conter n registros armazenados. Então, as primeiras colisões n em dado endereço hash p serão armazenadas na página p e uma busca linear através

dessas colisões estará totalmente contida nesta página. Entretanto a próxima colisão - isto é, a de número  $(n + 1)$  - deverá ser armazenada em página extra, e será necessária outra E/S.

Uma outra abordagem ao problema da colisão, talvez mais encontrada na prática, seria de tratar o resultado a partir da função hash, digamos  $a$ , como endereço de armazenamento, não do registro de dados, mas do "ponto âncora". O ponto âncora no endereço de armazenamento  $a$  é então considerado a cabeça de uma cadeia de ponteiros (uma cadeia de colisões), unindo todos os registros - ou todas as páginas de registros - que colidem em  $a$ . As colisões dentro de uma determinada cadeia de colisões serão mantidas em uma seqüência de campo hash, a fim de simplificarem as buscas subseqüentes.

### Acesso Hash Extensível

Uma outra desvantagem, além daquela descrita acima, é a seguinte: como o tamanho do arquivo de acesso hash cresce, o número de colisões também tende a crescer e, conseqüentemente, o tempo de acesso médio aumenta de modo correspondente (porque se gasta cada vez mais tempo na busca através dos conjuntos de colisões). O acesso hash extensível é uma variação interessante da técnica básica, que minoriza este problema. O acesso hash extensível, de fato, garante que a quantidade necessária de acessos em disco para localizar um registro específico (isto é, o registro com o valor específico da chave primária) nunca será maior do que duas e, normalmente, de apenas uma.

Obs.: Os valores do campo hash devem ser únicos no esquema de acesso hash extensível, onde se encontrarão naturalmente, se aquele campo for, de fato, a chave primária, como sugerido no início desta seção.

O esquema funciona como segue.

1. A função hash básica é  $h$ , e o valor da chave primária de um registro específico  $r$  é  $k$ . O acesso hash a  $k$  - isto é, avaliando-se  $h(k)$ .  
- produz um valor  $k'$ , denominado de pseudochave de  $r$ . As pseudochaves não são interpretadas diretamente como endereços; ao contrário, elas conduzem aos locais de armazenamento de um modo indireto, como descrito abaixo.
2. O arquivo armazenado tem um diretório associado ao mesmo, também, armazenado no disco. O diretório consiste em um cabeçalho, que contém um valor  $d$ , denominado profundidade do diretório, em conjunto com  $2^o$  ponteiros, que são ponteiros para as páginas de dados, que contêm os registros armazenados reais (registros múltiplos por página). Um diretório de pro-

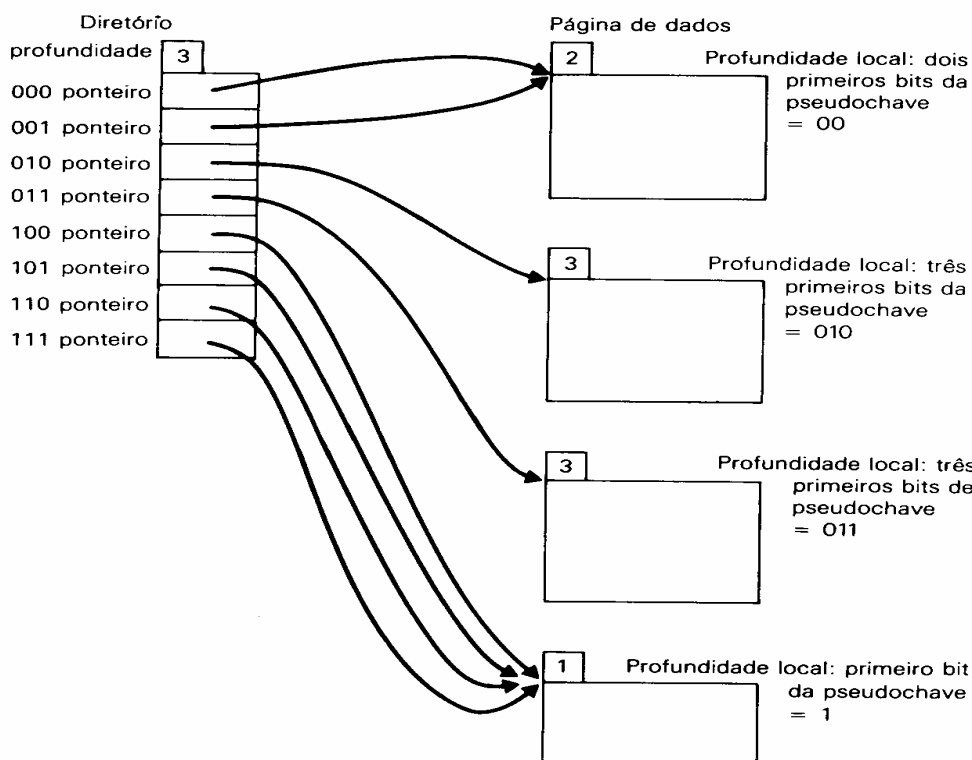
fundidade  $d$ , portanto, pode manejar um tamanho máximo de arquivo de  $2^d$  de páginas de dados distintas.

3. Se consideramos os bits-guia  $d$  de uma pseudochave como um inteiro binário sem sinal algébrico  $b$ , então o ponteiro de número  $i$  no diretório ( $1 \leq i \leq 2^d$ ) indica a página que contém todos os registros para os quais  $b$  toma o valor  $i - 1$ . Em outras palavras, o primeiro ponteiro indica a página que contém todos os registros para os quais  $b$  é apenas zero, o segundo ponteiro indica a página onde  $b$  é 0 ... 01, e assim por diante. (Esses  $2^d$  ponteiros são todos distintos, ou seja, existirão menos do que  $2^d$  páginas de dados distintos. Vide Fig. 12.) Deste modo, para que possamos achar o registro que tenha o valor  $k$  da chave primária, achamos  $k$  via acesso hash para descobrir a pseudochave  $K'$  e tirar os primeiros bits  $d$  da pseudochave; se esses bits tiverem o valor numérico  $i - 1$ , vamos até o ponteiro de nº  $i$  no diretório (primeiro acesso em disco) e o seguimos até a página que contém o registro desejado (segundo acesso em disco).

Obs.: O diretório será, na prática, pequeno o suficiente para ser mantido no armazenamento principal durante a maior parte de tempo, de modo que os "dois" acessos em disco serão normalmente reduzidos a um só na prática.

4. Cada página de dados também tem um cabeçalho, indicando a profundidade do local  $p$  da mesma ( $p \leq d$ ). Suponhamos, por exemplo, que  $d$  seja três e que o primeiro ponteiro no diretório (o ponteiro 000) aponte para a página onde a profundidade local  $p$  seja dois. A profundidade local dois significa, neste caso, que esta página não só contém todos os registros com pseudochaves iniciando em 000, mas que a mesma também contém todos os registros com pseudochaves iniciando em 001, 010 e 100.

5. Suponhamos que des

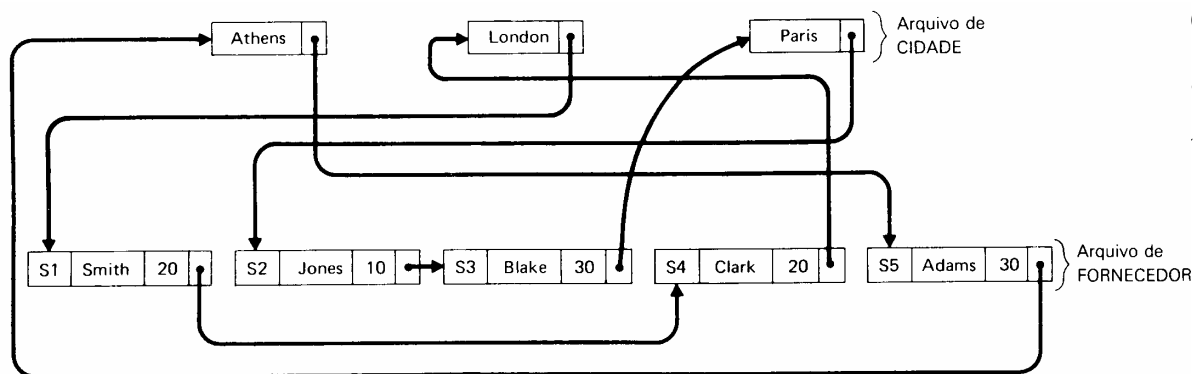


Neste ponto, a página será fracionada em duas; ou seja, adquire-se uma página nova e vazia, e todos os registros 001 são removidos da página antiga para a nova. O ponteiro 001 no diretório será modificado para apontar a nova página (o ponteiro 000 continua para a página antiga). A profundidade local p para cada uma das duas páginas será agora três, e não mais dois.

6. Continuando o nosso exemplo, suponhamos que a página de dados 000 esteja novamente cheia e tenha que ser fracionada. O diretório existente não pode manipular tal fracionamento, porque a profundidade local da página a ser fracionada já é igual à profundidade do diretório. Assim, "dobramos o diretório"; ou seja, aumentamos d por um e substituímos cada ponteiro por um par de ponteiros adjacentes idênticos. A página de dados agora pode ser fracionada; os registros 0000 continuam na página anterior, e os registros 0001 vão para a nova página; o primeiro ponteiro no diretório permanece inalterado (i.e., continua a apontar para a página anterior), o segundo ponteiro é modificado, a fim de apontar para a nova página. Observemos que a duplicação do diretório é uma operação barata, pois não necessita de acesso às páginas de dados.

#### 15.4 CADEIAS DE PONTEIROS

Suponhamos, novamente, que a consulta "Achar todos os fornecedores na Cidade C" seja importante. Uma outra representação armazenada que pode tratar razoavelmente bem esta consulta - possivelmente melhor do que um índice, embora de forma marginal - utiliza cadeias de ponteiros. Esta representação é ilustrada na Fig. 3.17. Como se pode ver, a mesma envolve dois arquivos armazenados, um arquivo de fornecedores e um arquivo de cidades, quase igual à representação do índice na Fig. 13 (desta vez ambos os arquivos encontram-se, provavelmente, no mesmo conjunto de páginas. Na representação de cadeias de ponteiros da Fig. 13, o arquivo de cidades, contudo, não é um índice, mas sim um arquivo-"pai" (ou fonte), como o mesmo é denominado. O arquivo de fornecedores é denominado consequentemente de arquivo-"filho", e a estrutura em si é um exemplo de uma "organização pai/filho".



campo cidade como tal foi removido do arquivo de fornecedores. O DBMS, para achar todos os fornecedores em Londres (digamos), pode pesquisar o arquivo de cidades pela entrada de Londres e, então, seguir a correspondente cadeia de ponteiros.

### **FIGURA 13 –Exemplo de uma estrutura pai/filho**

A principal vantagem da estrutura pai/filho (cadeia de ponteiros) é que os algoritmos inserir/anular são mais simples, e podemos ver que mais eficientes do que os algoritmos correspondentes para índice. Assim, a estrutura, provavelmente, deve ocupar menos armazenamento do que a estrutura de índices correspondente, porque cada valor da cidade aparece exatamente uma vez, e não de forma múltipla. As principais desvantagens são as seguintes:

- A única maneira de acessar o fornecedor de número  $n$  de uma determinada cidade é seguir a cadeia e acessar o 1º, 2º, ...,  $(n - 1)$  fornecedor também. Se os registros de fornecedor não forem agrupados de forma adequada, cada acesso envolvendo uma operação de pesquisa, o tempo que se gasta para acessar o fornecedor de número  $n$  pode ser bastante considerável.

- Embora a estrutura possa ser adequada para a consulta "Achar os fornecedores de uma determinada cidade", a mesma não é de nenhum auxílio - de fato, é um obstáculo - quanto se trata da consulta oposta "Achar a cidade de determinado fornecedor" (onde o fornecedor determinado é identificado por um número de fornecedor determinado). Para a última consulta, nem o acesso hash, nem um índice no arquivo de fornecedores bastará; observemos que uma estrutura pai/filho, com base nos números dos fornecedores não faria muito sentido (por que não?). E mesmo se o registro do dado fornecedor fosse localizado, ainda assim seria necessário seguir-se a cadeia até o registro-pai para descobrir a cidade desejada (a necessidade deste passo adicional é a nossa justificativa para reivindicar que a estrutura pai/filho é um obstáculo para esta classe de consultas).

Observemos, ainda, que o arquivo-pai (cidade) necessitará também de um acesso hash ou um índice, se o mesmo for de um tamanho expressivo. Conseqüentemente, as cadeias de ponteiros sozinhas não são, na realidade, uma base adequada para uma estrutura de armazenamento - outros mecanismos, tais como índices, também serão necessários.

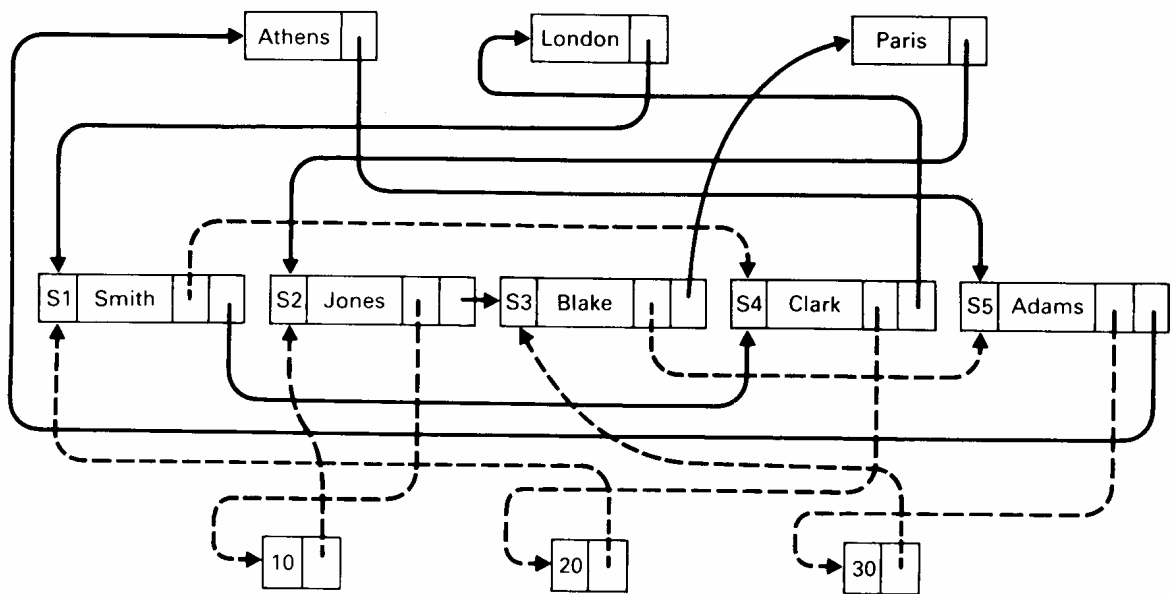
- Criar uma estrutura pai/filho para um conjunto existente de registros é uma tarefa incomum porque as cadeias de ponteiros passam através dos registros armazenados (isto é, os prefixos dos registros incluem fisicamente os ponteiros relevantes) e, também, porque os valores do campo relevante são decompostos para fora dos registros-filho e colocados, ao contrário, nos registros-pai. De fato, tal operação necessitará de uma organização do banco de dados, pelo menos para a parte relevante do mesmo.

A criação de um novo índice para um conjunto existente de registros, ao contrário, seria um quesito mais direto. (A criação de um novo acesso hash também implicará reorganização.)

A estrutura pai-filho básica possibilita diversas variáveis. Por exemplo:

- Os ponteiros podem ser feitos em sentido duplo. A vantagem é que tal variável simplifica o ajuste de ponteiro necessário para a operação de anulação de um registro-filho.
- Uma outra ampliação seria a inclusão de um ponteiro (um "ponteiro-pai" de cada registro-filho para o pai correspondente); isto reduziria à quantidade de passagens na cadeia que são necessárias para se responder à consulta "Achar a cidade para determinado fornecedor" (observemos, entretanto, que esta ampliação não afeta a necessidade de um acesso hash ou índice para auxiliar na resposta à consulta).
- Uma outra variável seria não remover o campo do arquivo de fornecedores, mas repetir o campo nos registros de fornecedores; certas possibilidades de recuperação (por exemplo, "Achar a cidade do fornecedor S4") torna-se-iam mais eficientes. Observemos, porém, que o aumento da eficiência não tem nada a ver com a estrutura da cadeia de ponteiros em si - e que ainda será provavelmente necessário um acesso hash ou índice nos números de fornecedores.

Finalmente, assim como é possível ter quaisquer quantidades de índices em um determinado arquivo armazenado, também é possível ter quaisquer quantidades de cadeias de ponteiros em determinado arquivo armazenado. (Ambos também são possíveis, embora incomuns na prática.) A Fig. 14 mostra uma representação de um arquivo de fornecedores que envolve duas cadeias de ponteiros distintas e, assim, duas estruturas-pai/filho distintas, uma com o arquivo de cidade como pai (como na Fig. 13) e uma com o arquivo de status como pai. O arquivo de fornecedores é o arquivo-filho para ambas as estruturas.



**FIGURA 14 – Exemplo de uma organização pai/filho**

Texto gentilmente cedido por **Marcos Mina Kamada** ([kamada@rio.com.br](mailto:kamada@rio.com.br))



[www.sti.com.br](http://www.sti.com.br)