

Organização de Arquivos de Dados

Objetivo: dada uma sequência de **INSERT**, **DELETE** e **UPDATE**, desenhar como fica o arquivo de dados de cada tabela: em que bloco e em que slot cada tupla mora (o **rid**), o que vai para a **free list** e quando uma tupla **migra**.

Comece aqui

A ideia em uma frase. O disco não lê um dado de cada vez: ele lê **páginas inteiras**, como um caderno em que você só consegue abrir uma página por vez. O banco guarda cada tabela num arquivo feito dessas páginas, chamadas **blocos**, todas do mesmo tamanho (ex.: 256 bytes). Cada linha da tabela ocupa um pedaço de algum bloco. Este cheat sheet ensina a responder: *depois desses comandos, em que bloco e em que posição ficou cada linha?*

O que a questão pede. Um desenho dos blocos do arquivo, com cada linha no lugar certo (**[chave, ...]**), os espaços vazios, a lista de espaços livres e o cabeçalho do arquivo.

Palavras que vão aparecer:

- **Tupla** (ou **registro**): uma linha da tabela.
- **Bloco** (ou **página**): um pedaço de tamanho fixo do arquivo. É a unidade que o disco lê.
- **Slot**: a posição de uma tupla dentro do bloco (1^a, 2^a, 3^a...).
- **rid** (*row id*): o endereço da tupla, formado pelo número do bloco + número do slot. **0302** é o bloco 03, slot 02.
- **Free list**: a lista dos buracos que os **DELETE** deixaram, no formato **<rid, tamanho>**, para reaproveitar depois.
- **PCTFREE**: a porcentagem de cada bloco que o **INSERT** deixa vazia, para as tuplas crescerem num **UPDATE** futuro.
- **Migração**: quando um **UPDATE** aumenta a tupla e ela não cabe mais no bloco. Ela muda de lugar e de rid.

arquivo da tabela product (blocos de 256 B)

bloco 01:	[W10]	[AZA]	[NB2]	[WRF]	[MSP]	← 5 tuplas, sobram 2 B
	0101	0102	0103	0104	0105	← rid = bloco + slot
bloco 02:	[WRP]	[AMP]	[WFL]	[RSR]	[GN2]	
	0201	0202	0203	0204	0205	

Onde este arquivo se encaixa. É o **1º de 3**. Os rids que saem daqui são exatamente o que o índice guarda nas folhas ("Banco de Dados: Indexação e Árvore B+"). O número de blocos de cada tabela é a base de toda conta de custo ("Banco de Dados: Custo de Consultas").

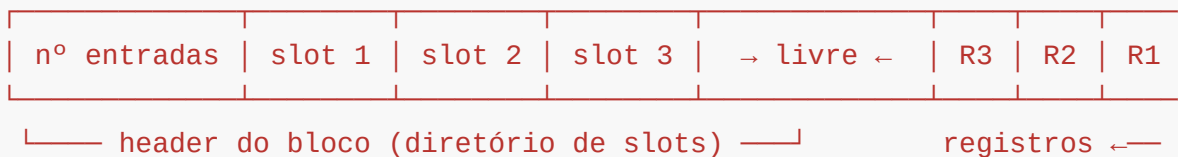
Se você está perdido, leia nesta ordem: este "Comece aqui", depois o **Exemplo resolvido (seção 8)** acompanhando a **Receita (seção 7)** passo a passo. As seções 1 a 6 são de consulta: volte a elas quando uma conta do exemplo não fizer sentido.



1. Conceitos de apoio

Conceito	O que é	Por que importa no exercício
Bloco (página)	Unidade de transferência entre disco e memória. Para ler 1 byte, lê-se o bloco inteiro.	Todo custo é contado em blocos; todo arquivo é uma sequência de blocos.
Registro	Sequência de campos: uma tupla.	O tamanho dele decide em que bloco ele cabe.
Fixo vs variável	Fixo: só campos de tamanho fixo (INT , CHAR). Variável: tem VARCHAR , campo opcional ou repetido.	Com VARCHAR , cada tupla tem um tamanho diferente.
Unspanned	O registro não atravessa a fronteira do bloco.	Se não cabe inteiro, vai para outro bloco, e a sobra fica vazia.
Spanned	O registro continua em outro bloco, ligado por ponteiro.	Só quando o enunciado permitir.
Fator de bloco (Bfr)	Registros por bloco: $Bfr = \lfloor B/R \rfloor$.	Com ele, $b = \lceil r/Bfr \rceil$ blocos.
Header do arquivo	Descritor do arquivo: número de blocos, endereços, formato.	Entra no diagrama da resposta.
Orientado a linha	A tupla inteira fica junta no bloco (o oposto é o armazenamento colunar).	O "registro" é a linha toda.

Estrutura do bloco (slotted page). O bloco começa com um header que guarda o número de entradas e um **diretório de slots** (posição e tamanho de cada registro). Os registros crescem do fim para o começo, e o espaço livre fica no meio. O rid aponta para o *slot*, então mover um registro dentro do próprio bloco não muda o rid.



⚠ Se o enunciado não der o tamanho do header do bloco, **não desconte nada**: o bloco inteiro é espaço útil.

Organizações de arquivo

Organização	Inserção	Busca	Observação
Heap (não ordenado)	Barata: em qualquer lugar com espaço	Linear (lê tudo)	É o caso do exercício: a política (first-fit etc.) decide o bloco.
Ordenado (sequencial)	Cara: mantém a ordem, reorganiza	Busca binária	Ler em ordem da chave é muito eficiente.

Organização	Inserção	Busca	Observação
Hash	Função hash do campo → bloco	Ótima só para igualdade	Colisões: endereçamento aberto, encadeamento, hash múltiplo.

Remoção: ou se reescreve o bloco (compactando), ou se usa **marcador de remoção + free list**. No segundo caso, o buraco fica onde está e é reaproveitado depois. É o modelo do exercício.



2. Tamanho de cada registro

O tamanho do registro é a soma dos campos: $R = \sum \text{campos}$.

Tipo	Ocupa	Exemplo (2 bytes por caractere)
INT	O que o enunciado disser	8 B
CHAR(n)	Sempre $n \times$ bytes/caractere, mesmo com texto menor	CHAR(10) com 'Nike' = 20 B
VARCHAR(n)	Só os caracteres usados $\times$ bytes/caractere	'Winflo 10' = $9 \times 2 = 18$ B
Espaço, hífen	Contam como caractere	'Gel-Nimbus' = 10 caracteres

- 1 2 3 4 Um valor **'1'** (texto) numa coluna **INT** é convertido e continua ocupando o tamanho do **INT**.
- 1 2 3 4 **DEFAULT nextval('seq')** numera na ordem dos **INSERT**: 1, 2, 3, ...
- Prefixo de tamanho do **VARCHAR** e header de bloco só entram na conta quando o enunciado os informa.



3. PCTFREE: reserva para UPDATE

$$\text{limite do INSERT} = \left\lfloor B \times \left(1 - \frac{\text{PCTFREE}}{100} \right) \right\rfloor$$

- PCTFREE 5** com bloco de 256 B: $\lfloor 256 \times 0,95 \rfloor = \lfloor 243,2 \rfloor = 243$ B para **INSERT**. Os 13 B restantes (5% de 256) ficam reservados.
- PCTFREE 0**: o **INSERT** pode usar os 256 B.
- O **INSERT** ocupa só até o limite. O **UPDATE** pode usar a reserva, até encher o bloco (B).



4. Em que bloco o INSERT cai

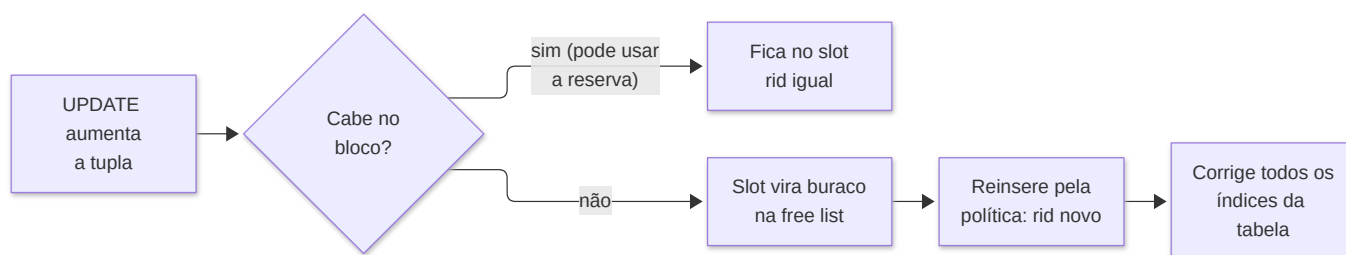
Política	Regra
First-fit	Percorre os blocos a partir do 01 e usa o primeiro onde a tupla cabe: um buraco da free list (em ordem de rid) com tamanho $\geq$ tupla, ou o espaço livre do bloco, respeitando o limite do PCTFREE.
Best-fit	Usa o lugar onde sobra menos espaço.

Política	Regra
Worst-fit	Usa o lugar onde sobra mais espaço.
Next-fit	Como o first-fit, mas continua de onde parou a última inserção.
Encher o último bloco	Só olha o último bloco; se não cabe, abre outro. Não reaproveita espaço anterior.

- Nenhum bloco serve → abre um bloco novo no fim do arquivo, e o header ganha +1 bloco.
- ⚠ No first-fit, uma tupla inserida depois pode cair num bloco **anterior**, se ainda houver espaço lá. É aqui que a resposta muda de uma política para outra.
- ⚠ Tupla maior que o buraco, mas que caberia somando o buraco ao resto do espaço livre do bloco: isso depende de compactar o bloco. Diga o que assumiu.

5. Efeito de cada operação

Operação	O que acontece no arquivo	rid	Índices
INSERT	Entra pela política (first-fit), até o limite do PCTFREE.	Novo	Ganham uma entrada.
DELETE	A tupla sai e ninguém se move : o buraco vira <rid, tamanho> na free list, ordenada por rid.	Deixa de existir	Perdem a entrada.
UPDATE que cabe	Cresce no lugar, podendo usar a reserva do PCTFREE.	Igual	Nada muda (se a chave não mudou).
UPDATE que não cabe	Migra : o slot antigo vira buraco na free list e a tupla é reinserida pela política.	Muda	Todos os índices da tabela são corrigidos.



- 🔗 A chave estrangeira de **outras** tabelas não muda: ela guarda o **valor** (ex.: **brand_id = 1**), não o endereço.
- 📌 No modelo deste exercício, o rid muda e os índices são corrigidos. SGBDs como o Oracle mantêm o rid deixando um ponteiro de encaminhamento no slot antigo (*row migration*). Use essa variante só se o enunciado pedir.

6. Como desenhar a resposta

- rid no formato PPRR**: **PP** é o bloco e **RR** o slot dentro do bloco, ambos a partir de 01. **0302** é o 2º registro do 3º bloco.

- Cada tupla é **[chave, ...]** , e cada buraco é **(livre, N B)** . Anote a ocupação de cada bloco, o header e a free list.

```
<tabela>    header: N blocos      free list: <PPRR, tamanho> -> ...  
└─ bloco 01 ────────────────────┘  
| [chave, ...] [chave, ...] (livre, 46 B) [chave, ...] | usado/B  
└─ bloco 02 ────────────────────┘  
| [chave, ...] [chave, ...]                        | usado/B  
└────────────────────────────────┘
```

✓ 7. Receita na prova

1. **Tamanho de cada tupla:** some os campos (**CHAR** cheio, **VARCHAR** pelo uso, × bytes por caractere).
Faça uma tabela com uma linha por tupla, incluindo o tamanho **depois** de cada **UPDATE** .
2. **Limite do INSERT** de cada tabela, pelo PCTFREE (arredonde para baixo).
3. **Insira uma por uma** pela política do enunciado, anotando a ocupação de cada bloco e o rid.
4. **DELETE:** apague sem mover ninguém e anote **<rid, tamanho>** na free list, em ordem de rid.
5. **INSERT depois de DELETE:** confira os buracos da free list antes do fim dos blocos (o first-fit percorre na ordem).
6. **UPDATE:** veja se cabe no bloco usando a reserva. Se não cabe, migre, mude o rid e diga que os índices são corrigidos.
7. **Desenhe** o arquivo final: header, free list e blocos com ocupação. Escreva a política que aplicou.



8. Exemplo resolvido completo

Enunciado (resumido). Bloco de 256 B, **INT** de 8 B, charset de 2 bytes por caractere, registros não extrapolam blocos, armazenamento orientado a linha. Política de (re)uso de blocos: **first-fit**. Free list com **<rid, tamanho>** ordenada por rid. O header do arquivo guarda o número de blocos.

```
CREATE TABLE brand (id INT DEFAULT nextval('brand_id_seq'), name CHAR(10),
                    description VARCHAR(100), ...) PCTFREE 5;
CREATE TABLE product (id CHAR(3), name VARCHAR(50), brand_id INT,
                      sold_units INT, ...) PCTFREE 0;
INSERT INTO brand ... 'Nike' (50 chars), 'Asics' (80), 'Mizuno' (40)
INSERT INTO product ... W10, AZA, NB2, WRF, MSP, WRP, RV6, WFL, RSR, GN2,
                      RV7, GE2, CR3 -- nesta ordem
DELETE FROM product WHERE id IN ('RV6', 'RV7');
INSERT INTO product ... ('AMP', 'Air Max Plus', '1', 1200);
UPDATE brand SET description = <75 chars> WHERE name = 'Nike';
```

Passo 1. Tamanhos.

- **product** : **CHAR(3)** 6 + **INT** 8 + **INT** 8 = 22 fixos, mais o nome × 2.
- **brand** : **INT** 8 + **CHAR(10)** 20 = 28 fixos, mais a descrição × 2.

id	name	Caracteres	Tamanho
W10	Winflo 10	9	40 B
AZA	Air Zoom Arcadia	16	54 B
NB2	Novablast	9	40 B
WRF	Wave Rebellion Flash	19	60 B
MSP	Metaspeed Sky Paris	19	60 B
WRP	Wave Rebellion Pro	17	56 B
RV6	Revolution 6	12	46 B
WFL	Wave Falcon	11	44 B
RSR	React Scape Run	15	52 B
GN2	Gel-Nimbus	10	42 B
RV7	Revolution 7	12	46 B
GE2	Gel-Excite	10	42 B
CR3	Cool Ride	9	40 B
AMP	Air Max Plus	12	46 B


```
brand    header: 3 blocos    free list: <0101, 128>
┌ bloco 01 ───────────┐
│ (livre, 128 B) [3, ...] │ 108/256 ← slot 0101 era da Nike
└ bloco 02 ───────────┘
│ [2, ...] │ 188/256
└ bloco 03 ───────────┘
│ [1, ...] │ 178/256 ← Nike migrada pelo UPDATE
└──────────────────┘
```

-  **Leitura adotada para brand**: first-fit ao pé da letra (cada INSERT procura o primeiro bloco com espaço). Com "encher o último bloco", a Mizuno iria para um bloco 03 próprio, e a Nike cresceria no lugar ($128 + 50 = 178$ cabe em 243). Escreva na prova a regra que aplicou.
-  **Header de bloco não descontado**, porque o enunciado não dá o tamanho dele.
-  RV6, RV7 e AMP têm exatamente 46 B: o exercício foi montado para o AMP cair no buraco do RV6.

! 9. Pegadinhas

- **CHAR(n)** ocupa *n* caracteres **sempre**; **VARCHAR** ocupa só o que foi usado.
- Conte espaço e hífen. Confira os nomes letra a letra: um caractere a mais muda o bloco.
- PCTFREE vale para o **INSERT**. O **UPDATE** usa a reserva. Arredonde o limite **para baixo**.
- First-fit volta aos blocos do começo: uma tupla nova pode cair num bloco antigo.
- **DELETE** não compacta nem move ninguém; só cria buraco na free list.
- **UPDATE** que não cabe **migra**: o rid muda, e todos os índices da tabela são corrigidos. A FK de outra tabela não muda.
- O arquivo não encolhe: um bloco esvaziado continua contado no header.

10. Variações que podem cair

Se o enunciado disser...	Faça
Best-fit / worst-fit / next-fit	Troque a regra do passo 3 (tabela da seção 4).
Header de bloco de h bytes	Espaço útil = $B - h$; o PCTFREE incide sobre o útil (diga o que assumiu).
VARCHAR com prefixo de tamanho	Some o prefixo (ex.: +2 B) em cada campo VARCHAR .
Registros spanned	A tupla pode continuar no bloco seguinte, então não sobra espaço no fim.
Charset de 1 byte por caractere	Refaça o passo 1 com $\times 1$.
Arquivo ordenado pela chave	A tupla vai para a posição da chave, não para o primeiro espaço livre.
UPDATE que diminui a tupla	Fica no lugar; o espaço liberado volta a ser livre no bloco.

