Índices

1. Listar as datas de consulta de um doente:

**select** data **from** consulta **where** num\_doente = <um\_valor>

Sabendo que os índices hash são ideais para seleção por igualdade, criamos um índice para organizar a coluna “num\_doente” da tabela “consulta” através de uma Hash Table, o que facilita e torna mais eficiente a comparação do “num\_doente” com o valor dado. Apesar de a chave primária da tabela consulta ser composta pelo “num\_cedula”, “num\_doente” e “data”, apenas o atributo “num\_doente” é usado na seleção por igualdade após o **WHERE**, pelo que a função de dispersão (hash) recebe apenas o parâmetro “num\_doente”.

1. Considere que há apenas seis especialidades: “E1” a “E6”. Pretende-se saber quantos médicos existem de cada especialidade.

**select** count(\*) **from** medico **where** especialidade = “Ei”

em que Ei é uma das seis especialidades.

Neste caso, apesar de também ser uma comparação com um valor dado, consideramos que deve ser utilizado um índice bitmap sobre o atributo “especialidade”, dado ao pequeno número de valores distintos para este atributo (apenas 6) em comparação com o número de rows na tabela “medico” (que ultrapassa em várias ordens de grandeza a memória disponível).

A tabela “medico” tem como chave primária o “num\_cedula”, porém este atributo não é relevante para a procura que nos é solicitada. Para além disso, como consideramos que o atributo “especialidade” não está sujeito a alterações constantes, a indexação através da criação de bitmaps para cada valor Ei do atributo “especialidade”, em princípio, não criará problemas e é a que faz mais sentido.

1. Nomes dos médicos de uma determinada especialidade. Para a resolução desta alínea considere, para além do referido sobre a dimensão das tabelas, os seguintes aspetos:
2. Os blocos do disco são de 2K bytes e cada registo na tabela ocupa 1K bytes.
3. Os médicos estão uniformemente distribuídos pelas 6 especialidades.

**select** nome **from** medico **where** especialidade = ‘Ei’

em que Ei é uma das seis especialidades.

Neste caso, vai ser necessário a criação de um índice através de uma B+ Tree desagrupado em que a chave de pesquisa é o atributo “nome” da tabela “medico”. Isto obriga a uma leitura máxima de log(n/2) (N). Como cada bloco do disco ocupa 2kB e cada registo ocupa 1kB, podemos então deferir que n≈1024 e, se existir um milhão valores de chave, acede-se apenas a log512(1000000)≈ 2 nós. Ou seja, um índice B+ permite-nos encontrar qualquer chave “nome” de uma forma muito mais eficiente. Resumindo, para blocos muito grandes o B+ Tree mostra-se bastante eficiente.

E um índice Hash denso e desagrupado para a chave “especialidade” da tabela “medico” permite-nos obter mais rapidamente os dados a partir da procura na Hash table, mais eficiente do que a B+ neste caso porque no **WHERE** é feita uma seleção por igualdade.

1. Listar os nomes dos médicos que deram consultas entre duas datas.

**select** nome **from** medico, consulta

**where** consulta.num\_celula=medico.num\_celula AND

consulta.data BETWEEN ‘data\_1’ AND ‘data\_2’

em que ‘data\_1’ e ‘data\_2’ são duas datas.

Para a seleção de igualdade do atributo “num\_cedula” nas tabelas “consulta” e “medico” é usada uma Hash Table na qual passamos como parâmetro para a função de dispersão o atributo “num\_cedula” pois estas são ideais para as situações de seleção de igualdade, como é o caso.

Seguidamente, consideramos que faz sentido indexarmos também por B+ Trees em relação ao atributo “data” da tabela “consulta” devido ao facto de estarmos à procura de um intervalo entre duas datas e o B+ Tree permitir percorrer facilmente e com rapidez um intervalo de valores do tipo “date” (que pode ser classificado segundo uma ordem).

Nesta situação, o B+ Tree consegue percorrer o intervalo de valores dado para o atributo “consulta.data” visto que existem ligações por ponteiros ao nível das folhas (o que, por exemplo, não se verifica na B Tree). Isto permite ainda que, partindo da raíz, seja possível chegar a todas as folhas.

Não faz sentido usar a chave primária da tabela consulta pois esta é composta pelo “num\_cedula”, “num\_doente” e “data” e o atributo “num\_doente” não é necessário para a indexação, pelo que criamos uma nova chave composta apenas pelos atributos “num\_cedula” e “data”.

**NOTA:** No schema.sql da entrega 3, o atributo “data” da tabela consulta foi alterado por nós para “data\_consulta”, visto que a palavra data estava a gerar problemas quando corríamos no postgresql.