Propostas de Soluções Testes de Sistemas de Bases de Dados 2014-15

Teste 1

1a) O ideal seria utilizar um índice secundário (non-clustered) em árvore B+ pois permitiria efetuar eficientemente pesquisas por nome da empresa (apesar do número elevado de tuplos), podendo também ser facilmente mantido. Uma das vantagens é que para além da igualdade suportaria também outros tipos de consultas (por exemplo, >=, LIKE, etc).

Um índice hash permitiria efetuar eficientemente pesquisas por igualdade de nomes mas obrigaria a ter um índice de hash extensível para poder suportar a evolução constante da base de dados. FALTA.

1b) É uma boa opção pois permitiria efetuar eficientemente pesquisas por código com apenas uma operação de seek e outra de leitura. Como não são necessárias muito atualizações e a tabela é pequena, o índice pode ser reindexado quando se degradasse a performance.

1c) Não haveria grande vantagem pois é muito natural que fosse sempre necessário ler a maioria dos blocos (com padrões de acesso aleatório ao disco) sendo assim preferível um *full table scan*.

1d) Não é indiferente. Como é natural pretender-se efectuar pesquisas por NIF o mais adequado seria (NIFEmpr,CAE) em vez de (CAE,NIFEmpr), pois na primeira situação temos o índice primeiro ordenado por NIF e depois por CAE. A ordem (CAE,NIFEMpr) obrigaria percorrer todo o índice para encontrar o número de identificaçãoo fiscal (logo seria preferível fazer um full table scan). A ordem (CAE,NIFEmpr) é mais adequada para obter as empresas com um determinado CAE.

1e) SELECT COUNT(*) FROM Empresas WHERE tipo = ...

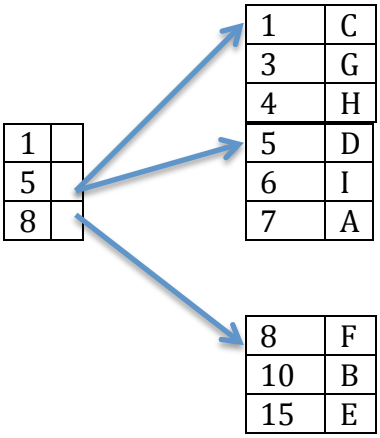
1f) Sim, seria benéfica pois permitiria obter a informação com um único acesso a disco. Em cada bloco seria colocado cada empresa seguida da informação correspondente das suas atividades. A tabela de codigosAtividade **NÃO** seria colocada nesse bloco pois resultaria em redundância de informação.

Exemplo:

Empresas(1,'XPTO',1,'2015-01-01')
    Atividades(1,1,TRUE)
    Atividades(1,2,FALSE)
Empresas(1000,'XPTO',1,'2015-01-01')
    Atividades(1000,3,TRUE)
...

2 a) 4 * 4 * 3 = 48

2b)



2c) Não existindo buckets cheios, a dispersão estática é mais vantajosa pois permite realizar todas as operações com um menor número de acessos a disco (e.g. pesquisa só necessita de 1). A dispersão extensível obriga sempre a pelo menos 2 acessos a disco.

Teste 2

1a)

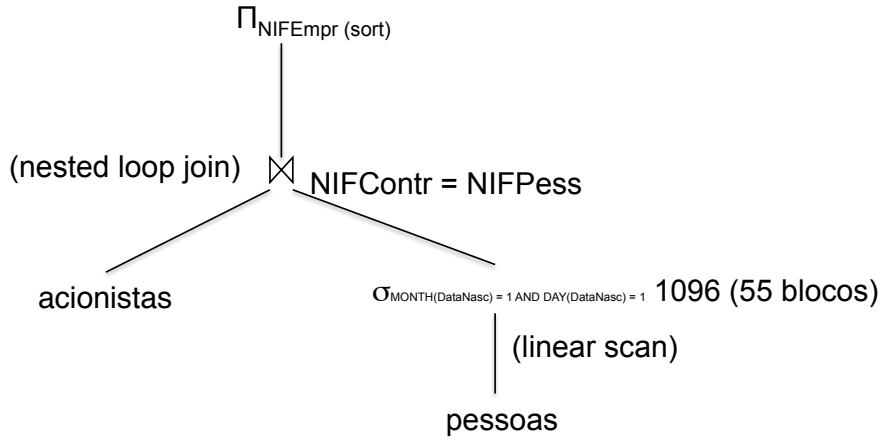
i) $n_{\sigma_{NOT(Primário=TRUE)}(atividades)} = n_{atividades} - n_{\sigma_{(Primário=TRUE)}(atividades)} = 350000 - 100000 = 250000$

A seletividade é 250000/350000 = 71,4%

ii) $n_{\sigma_{NOT(Primário=TRUE)}(atividades)} = n_{\sigma_{Primário=FALSE}(atividades)} = \frac{n_{atividades}}{V(Primário,atividades)} = \frac{350000}{2} = 175000$

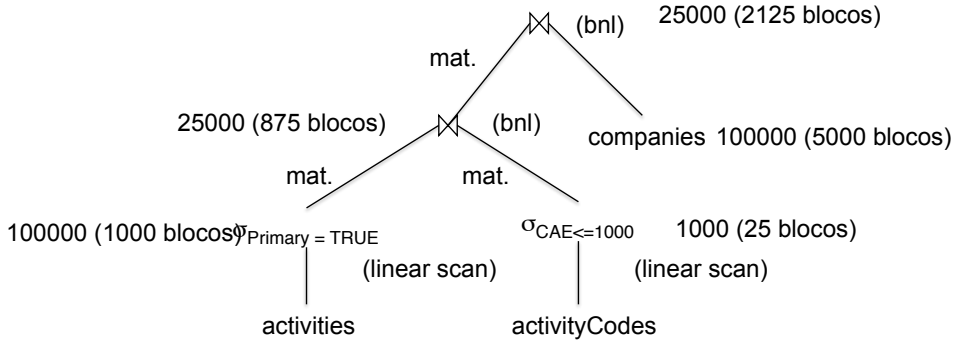
Logo a seletividade é 175000/350000 = 50%

1b) O plano mais eficiente seria:



A seleção sobre a tabela pessoas estima-se que devolva cerca de 1096 tuplos correspondendo a 400000/365 (assumindo que um ano tem 365 dias e que os nascimentos estão distribuídos uniformemente). Repare-se que o resultado desta seleção cabe integralmente em memória (no máximo ocupa 55 blocos) logo esta relação deverá ser a inner (convenção, fica do lado direito da junção) enquanto que a relação acionistas será a outer.

1c) O plano devidamente anotado com o número de tuplos e blocos correspondentes pode ser encontrado na figura seguinte.



A seleção sobre a tabela actividades devolve 100000 tuplos pois todas as empresas têm código primário. Esta informação ocupa 1000 blocos. A seleção sobre a tabela atividades devolve exactamente 1000 tuplos, ocupando 25 blocos. A junção destas duas tabelas estima-se que devolva 25000 tuplos pois correspondem a 25% das atividades. Cada tuplo tem informação de uma atividade e de um código, ou seja iremos necessitar de 25000/100 + 25000/40 = 875 blocos. Assim, verificamos que para a junção na raiz nenhum dos resultados cabe em memória, devendo a inner relation ser a tabela empresas no block neste loop join. Vamos então determinar o custo de utilizar o block nested loop join:

custo bnl = $(875*5000 + 875)*t_R + 2*875*t_S = 4375875t_R+2*875*10* t_R = 4393375t_R$

O único índice disponível é o da chave primária da tabela empresas. Este índice tem altura 3 (100*100*99).

custo inl = $875*(t_R + t_S)+25000*(3+1)*(t_R + t_S) = 100000*(t_R + t_S) = 100875*11*t_R = 1109625*t_R$

Logo, compensa a utilização do indexed nested loop join.

1d) A técnica não pode ser utilizada nesta query pois é sempre necessário verificar que uma empresa é acionista de outra (o que pode não acontecer) obrigando a efetuar a junção das duas tabelas.

1e) Qualquer condição de junção que envolva atributos de duas tabelas distintas e que não seja uma equijunção (equijoin). Exemplo:

```
SELECT * FROM Empresas, Pessoas
WHERE DataInício > Pessoas.DataNasc;
```

2a) O custo é $\lceil b_r / (M-2) \rceil * b_s + b_r$ block transfers $+ 2 \lceil b_r / (M-2) \rceil$ seeks

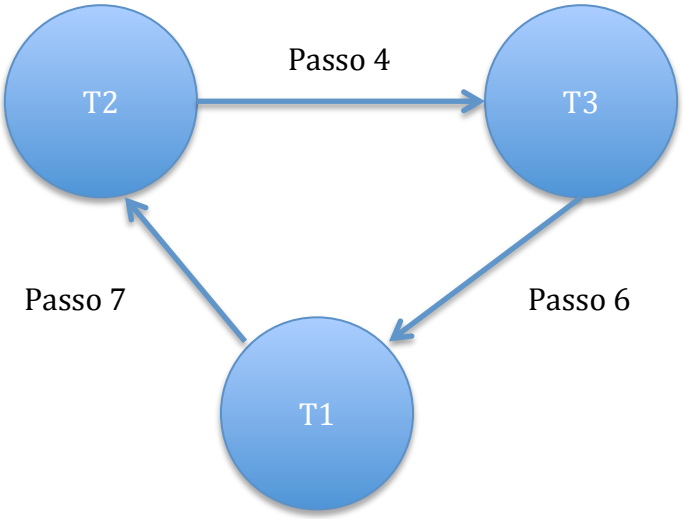
O factor $\lceil b_r / (M-2) \rceil$ é o número de iterações do ciclo exterior, tendo-se ler todos os blocos de s, logo a componente $\lceil b_r / (M-2) \rceil * b_s$. Temos de ler também sempre todos os blocos da relação exterior, daí somar-se $+ b_r$ transferências de blocos. Repare-se que para cada iteração do ciclo exterior é necessário fazer um seek para a relação exterior e outra para a relação interior, daí o termo $2 \lceil b_r / (M-2) \rceil$ seeks.

2b) 10 * 20 = 200 processadores (ver slides)

2c) Uma das funções de dispersão é para efetuar o particionamento das relações originais. A outra é para a construção da tabela de dispersão em memória (build) partição a partição para depois cruzar com a probe. Se ambas as funções de dispersão forem as mesmas há maior risco de colisões nesta segunda fase pois todos os tuplos pertencem ao mesmo bucket da primeira função(num caso extremo todos os tuplos podem ficar no mesmo bucket).

Teste 3

1a) Sim, o escalonamento provoca um deadlock no passo 7. A situação poderia ser detectada recorrendo a um grafo de dependências (wait-for) verificando a existência de um ciclo com a introdução da dependência entre T1 e T2 (T1 fica à espera de T2).



1b)

Passo	T1	T2
1	lock-x(A)	
2	write(A)	
3	unlock(A)	
4		lock-x(A)
5		read(A)
6		A := A + 1
7		write(A)
8		unlock(A)
9	rollback	

A transação T2 efetua a leitura do valor escrito por T1 que entretanto faz o rollback. A transação T2 também deve ser anulada pois caso contrário estaria a ler um valor que já não existe na BD. Repare-se que ambas as transações seguem o protocolo de commit em duas fases (fase de aquisição de locks, seguida de fase de libertação de locks).

1c) O log vai conter adicionalmente os seguintes registos:

<T2,B,1>
<T2 abort>

Após recuperação os valores de A, B e C serão 1, 1 e 4 respectivamente. Vejamos como. As operações estão numeradas por ordem de execução:

Log	Redo	Undo
<T1 **start**>		
<T2 **start**>		Escreve <T2 abort> no log undolist = {}
<T1,A,1,2>		
<**checkpoint** {T1,T2}>	Redo começa a partir daqui undolist = {T1,T2}	
<T2, B, 1, 3>	B := 3	B := 1 Escreve <T2,B,1> no log
<T3 **start**>	undolist = {T1,T2,T3}	
<T1,A,1>	A := 1	
<T3,C,1,4>	C := 4	
<T3 **commit**>	undolist = {T1,T2}	
<T1 **abort**>	undolist = {T2}	

Repare-se que na fase de redo o log é percorrido do início para o fim, enquanto que na fase de undo é percorrido do fim para o ínicio até a undolist ficar vazia.

1d) Os valores escritos são 0, 1 e 2 nos passos 2, 6 e 9 respectivamente. O escalonamento é serializável e corresponde à execução de T2 primeiro e T1 depois (<T2,T1>).

1e) Assuma-se que existem três transações T1, T2 e T3 a pretender atualizar simultaneamente o registo de um mesmo contribuinte. Imagine-se que T1 obtém locks nas réplicas 1 e 2, T2 nas réplicas 3 e 4, e T3 na réplica 5. Nenhuma das transações tem 3 réplicas para poder efetuar a atualização, ficando mutuamente bloqueadas. Repare-se que num sistema centralizado uma das transações conseguiria obter o lock enquanto que as outras ficariam à espera, não ocorrendo bloqueamento mútuo.

2 a) A técnica de Write-Ahead Logging permite utilizar buffers para os registos de log melhorando a performance do sistema pois evita a escrita para armazenamento estável após a execução de qualquer alteração à base de dados. Sucintamente, a regra WAL obriga à escrita dos registos de log em buffer antes da escrita dos blocos com a informação alterada (o que pode ocorrer bastante depois).

2b) O problema principal com o protocolo 2PC é que com a falha do coordenador da transação o sistema poderá ficar bloqueado à espera da sua recuperação. Esta situação ocorre quando os participantes têm registos <ready T> mas nenhum deles tem nem <abort T> nem <commit T>, logo têm de aguardar pela decisão que o coordenar possa já ter tomado.
O protocolo 3PC introduz uma nova fase em que o coordenador comunica a sua decisão a pelo menos K sites, evitando o problema quando menos de K sites falham. Contudo, assume-se que não podem ocorrer particionamentos de rede. O protocolo é mais pesado pois envolve mais troca de mensagens.