Sistemas de Bases de Dados
Resolução 2º teste 2015/16

1a) O atributo número é uma chave estrangeira em Viagens referenciando Carreiras. Logo, o número de tuplos retornados é igual ao número de tuplos retornados pela seleção em viagens:

$$n_{\sigma_{número \leq 10 \wedge data=2016-05-30}(Viagens \bowtie Carreiras)} = n_{\sigma_{número \leq 10 \wedge data=2016-05-30}(Viagens)}$$
$$= n_{Viagens} \times \frac{n_{\sigma_{número \leq 10}(Viagens)}}{n_{Viagens}} \times \frac{n_{\sigma_{data=2016-05-30}(Viagens)}}{n_{Viagens}}$$
$$= n_{Viagens} \times \frac{n_{Viagens} \times (10/12000)}{n_{Viagens}} \times \frac{n_{Viagens} \times (1/365.25)}{n_{Viagens}} =$$
$$= 3000000 \times \frac{10}{12000} \times \frac{1}{365.24} \cong 7$$

1b) O resultado do output da execução do escalonamento em modo READ UNCOMITTED é (0,2,2). Este output não é idêntico a uma possível execução em série das transações. Se for T1, T2 obtemos (1,1,2) e se for T2,T1 gera-se (0,1,2).

1c)

#	**Log**	**Memória**	**Disco**
			c[1].lorig=0 c[2].lorig=0 c[3].lorig=0
1	<T1,start>		
2	<T1,carreiras[1].locOrigem,0,1>	c[1].lorig=1	
3	<T2,start>		
4	<T2,carreiras[2].locOrigem,0,2>	c[2].lorig=2	
5	<T3,start>		
6	<T3,carreiras[3].locOrigem,0,2>	c[3].lorig=2	
7	<T1,commit>		
8	<checkpoint {T2,T3}>		c[1].lorig=1 c[2].lorig=2 c[3].lorig=2
9	<T3,carreiras[3].locOrigem,0>	c[3].lorig=0	
10	<T3,abort>		
11	**CRASH**		
12	**RECOVER**	c[3].lorig=0	
13	<T2,carreiras[2].locOrigem,0>	c[2].lorig=0	
14	<T2,abort>		

1d) Em modo Snapshot Isolation só há conflitos potencialmente entre escritas (em modo first-commiter-wins)

#	**Transação 1**	**Transação 2**
1		Begin transaction
2	Begin transaction	
3	UPDATE Carreiras SET localOrigem=1 WHERE número = 1;	
4	COMMIT;	
5		UPDATE Carreiras SET localOrigem=2 WHERE número = 1;
6		COMMIT; -> ERRO DE SERIALIZAÇÃO

No caso do Oracle que utiliza a estratégia first-updater-wins o erro é logo detetado no passo 5 mas precisa de locks para o implementar (não espera pelo COMMIT para verificar problemas de serialização)

1e)

Operação	Local 1	Local 2	Local 3	Resultado
read C(1,X,Y)	C(1,1,2):1	C(1,2,1):2	C(1,2,1):2	X= 2 , Y= 2
write C(1,3,1)	C(1,3,1):3	X	C(1,3,1):3	Sucesso
write C(1,3,2)	X	X	C(1,3,1):3	Insucesso

ATENÇÃO: EXISTE UMA GRALHA NO ENUNCIADO QUE ESTÁ NO CLIP (marcada a encarnado)

2a) As árvores profundas à esquerda (left-deep join trees) permitem ao SGBD implementar avaliação em pipeline e reduzir a complexidade do algoritmo de optimização de junções pois o espaço de procura é mais reduzido se bem que continue a ter complexidade exponencial.

2b) O algoritmo de 2 Phase Commit permite realizar atomicamente transações num ambiente distribuído. O algoritmo tem 2 fases:

1) Obter a decisão dos participantes, em que o coordenador adiciona o registo <prepare T> ao seu log e envia a mensagem de <prepare T> a todos os outros participantes onde T executou. Cada participante toma a decisão se é para abortar (juntando <no T> ao seu log ou se do ponto vista dele pode ser confirmada a transação (juntando <ready T> ao seu log), essa decisão é comunicada ao coordenador.
2) Regista a decisão: o coordenador recebe de todos os participantes se a transação é para confirmar ou não, tomando a decisão final juntando commit ou abort ao seu log. Depois do log ficar em armazenamento estável comunica a decisão aos restantes participantes para efetuar a ação apropriada no seu local.

2c) A utilização de checkpoints evita que em caso de falha todo o log seja percorrido para restaurar o estado correcto. O checkpoint realiza as seguintes operações:

1) Escreve todos os registos do log para armazenamento estável
2) Escreve todos os buffers em memória modificados para disco
3) Escreve no log o registo <checkpoint {T1,...,Tn}> em que T1,...,Tn são as transações em execução aquando da realização do checkpoint
4) Enquanto decorrer o checkpoint não se podem realizar operações que modifiquem dados na base de dados.