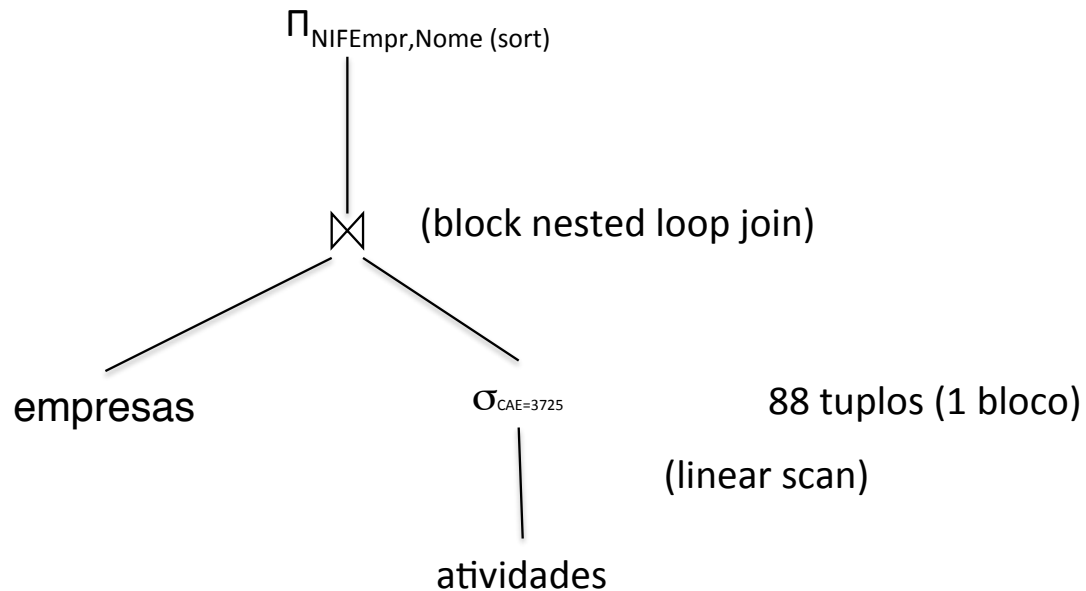
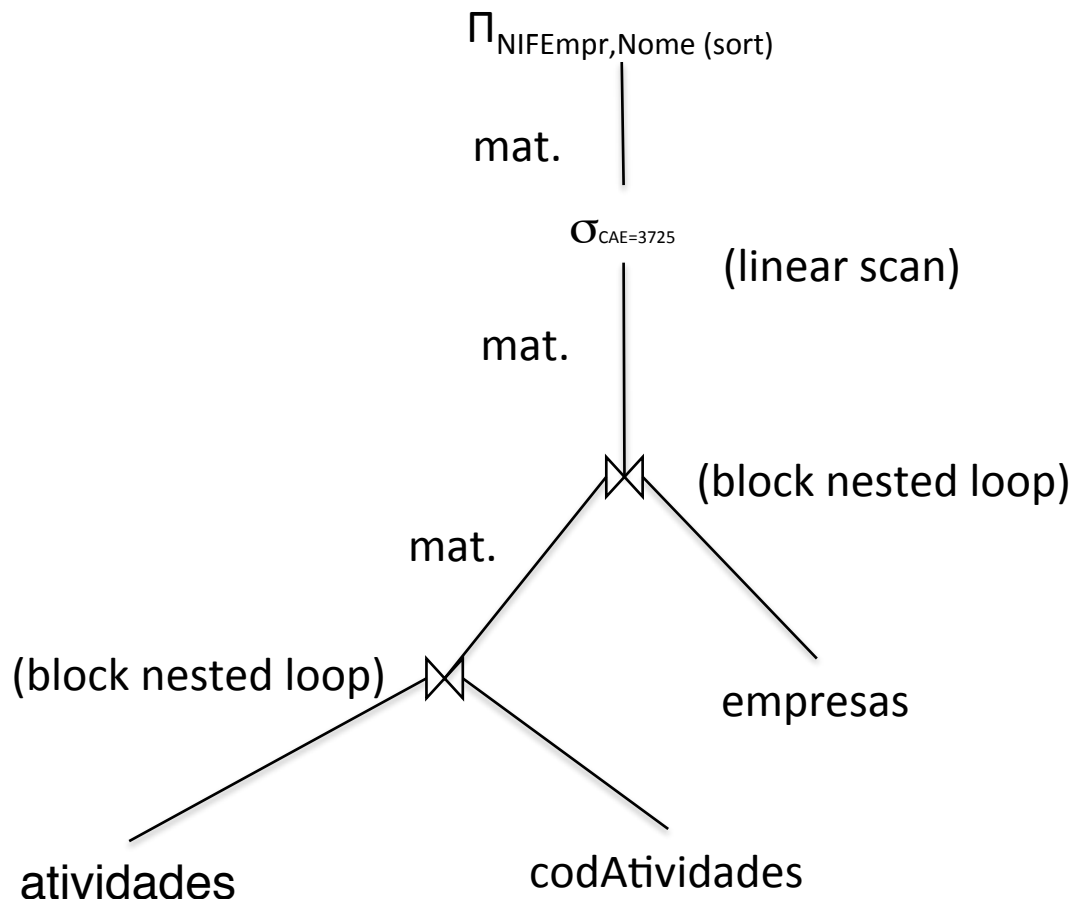


Proposta de Resolução do Exame de Recurso
Sistemas de Bases de Dados 2014/15

1a) O seguinte plano é muito eficiente pois fez-se uma minimização de junção e moveu-se a seleção para junto da tabela para minimizar o número de tuplos, que cabem em memória:



Outro plano (menos eficiente) pois envolve mais junções e a seleção é efetuada mais acima (ver a questão 1d para se ter uma ideia do custo...)



1b) O ideal sera criar índices bitmap pois permite efetuar contagens de forma muito eficiente. A cardinalidade do atributo tipo é pequena (4 valores possíveis) e da expressão YEAR(data_inicio) também não é muito grande (50 valores possíveis).

```
CREATE BITMAP INDEX tipoidx ON empresas(tipo);
CREATE BITMAP INDEX anoidx ON empresas(YEAR(data_inicio));
```

Repare-se que o espaço para armazenar o bitmap para um dos valores é cerca de 12500 byte (100.000/8) o que cabe normalmente em 2 a 4 blocos de disco.

1c) Uma pesquisa linear vai obrigar a transferir todos os blocos da tabela empresas para memória $100.000/20 = 5000$ blocos. Isto pode ser executado $5000 * t_T + 1 * t_S = 5000 * t_T + 10 * t_T = 5010 t_T$.

Sem informação adicional temos de supor uma distribuição uniforme dos valores. Logo:

$$n_{\sigma_{DataInicio=?}}(empresas) = \frac{n_{empresas}}{V(DataInicio, empresas)} = \frac{100000}{365,25 * 50} \cong 5,5$$

Ou seja, esperamos obter cerca de 6 tuplos. A altura da árvore B+ será 3 pois temos $100^2 < 100.000 < 100^3$ (para ser mais preciso, $100 * 99 < 100.000 < 100^2 * 99$). Assim, necessitamos de 3 transferências e 3 seeks para ler os blocos do índice e mais 6 transferências e 3 seeks para ler os blocos com os tuplos (o índice é secundário). Formalmente:

$$(h_i + n) * (t_T + t_S) = (3 + 6) * (t_T + t_S) = 9 * (t_T + 10 t_T) = 9 * 11 * t_T = 99 t_T.$$

(NOTA: com um pouco de cálculo verifica-se que para $n < 453$ compensa utilizar o índice).

1d) As tabelas envolvidas na query têm as seguintes dimensões em termos de número de blocos em disco:

Tabela	Número de Tuplos	Número de blocos
codAtividades	4000	$4000/50 = 80$
atividades	350.000	$350.000/100 = 3500$
empresas	100.000	$100.000/20 = 5000$

Há basicamente 3 hipóteses para efetuar as junções (já tendo em atenção quais as outer e as inner):

1. $(emp \bowtie_{codAt} at) \bowtie at$
2. $emp \bowtie (at \bowtie_{codAt})$
3. $(at \bowtie_{emp}) \bowtie codAt$

Podemos descartar imediatamente a 1ª hipótese pois corresponde a um produto cartesiano gerando $350.000 * 4000$ tuplos = 14E+8... Vamos ver os restantes dois casos (ignorando custos de materialização):

Operação	Tuplos	Blocos	Custo	Observação
(at ⋈ codAt)	350.000	3500+7000=10500	$(3500+80)t_T + 2t_S$	codAt cabe em memória
emp ⋈ (at ⋈ codAt)	350.000	10500+17500=28000	$(5000*10500+5000)t_T + 5000 t_S$	Emp menor

Operação	Tuplos	Blocos	Custo	Observação
(at ⋈ emp)	350.000	3500+17500=21000	$(3500*5000+3500)t_T + 3500t_S$	at menor
(at ⋈ emp) ⋈ codAt	350.000	17500+7000=28000	$(21000+80)t_T + 2t_S$	codAt cabe em memória

Repare-se que a última hipótese é cerca de três vezes mais eficiente. Logo a ordem preferível de junção é (at ⋈ emp) ⋈ codAt.

1e) O facto de existir índice primário obriga a que todos os tuplos estejam ordenados fisicamente pela chave primária. No caso dado, os tuplos estão ordenados pelo NIF da empresa e depois por NIFContribuinte. Logo, todos os tuplos de uma empresa estão armazenados consecutivamente em disco.

Assim, basta utilizar o índice para navegar para o primeiro tuplo $\geq A$. Seguidamente, percorrem-se sequencialmente os blocos do disco somando os valores do capital social. Quando se encontra um tuplo de uma nova empresa (ou se chegar ao fim), gera-se um tuplo para o output e coloca-se a soma a 0. Termina-se quando encontrarmos um tuplo com $NIF_{Empresa} > B$ ou se chegarmos ao final da relação.

1f) O log gerado é o seguinte

```

<T1 start>
<T1, empresas[1].tipo,0,1>
<T1, empresas[2].tipo,0,1>
<T2 start>
<T2, empresas[3].tipo,0,2>
<checkpoint {T1,T2}>
<T1, empresas[2].tipo,0>
<T1, empresas[1].tipo,0>
<T1 abort>
<T3 start>
<T3, empresas[1].tipo,0,1>
<T3 commit>

```

1g) Desde que haja a atualização de um mesmo tuplo em modo SERIALIZABE é gerada uma exceção pelo Oracle (exemplo):

	T1	T2
1	begin transaction UPDATE empresas SET Tipo = 1 WHERE NIFEmpr = 1;	
2		begin transaction UPDATE empresas SET Tipo = 2 WHERE NIFEmpr = 1;
3	COMMIT;	
4		COMMIT;

1h) Qualquer query que altere (quase) todos os tuplos de uma tabela é um bom exemplo.

```
UPDATE pessoas SET pessoas = UPPER(pessoas);
```

GRUPO 2

2a) Ver slide 12.18 (Aula 6)

2b) Ver slide 289 (Aula 8)

2c) Ver slides 19.41 a 19.43 (Aula 10)