**Relatório: 1º Mini-Projeto**

**Administração e Optimização de Bases de Dados**

|  |  |
| --- | --- |
| **70502:** João Carlos Duarte Santos Oliveira Violante – METI  **70599:** João Miguel Cordeiro Monteiro – METI  **70627:** Pedro Luís Galvão Raminhas – MEIC | **Grupo 2** |

**1. SQL Server Databases**

1. Comando T-SQL para criação da base de dados NutrientsDB:

|  |
| --- |
| CREATE DATABASE NutrientsDB ON  PRIMARY (  NAME='NutrientsDB\_Part1',  FILENAME='C:\Users\Public\NutrientsDB\_Part1.mdf',  SIZE=50MB,MAXSIZE=1GB,FILEGROWTH=5MB),  FILEGROUP NutrientsDB\_Part2 (  NAME='NutrientsDB\_Part2',  FILENAME ='C:\Users\Public\NutrientsDB\_Part2.ndf',  SIZE = 100MB,MAXSIZE=UNLIMITED,FILEGROWTH=50%),  FILEGROUP NutrientsDB\_Part3 (  NAME='NutrientsDB\_Part3',  FILENAME ='C:\Users\Public\NutrientsDB\_Part3.ndf',  SIZE = 50MB,MAXSIZE=UNLIMITED,FILEGROWTH=50%)  LOG ON (  NAME = 'NutrientsDB\_Log',  FILENAME = 'C:\Users\Public\Nutrients\_Log.ldf',  SIZE=25MB,MAXSIZE = 250MB,FILEGROWTH = 50%); |

1. Comando T-SQL para criar a tabela *Cheese* com as especificações indicadas no enunciado:

|  |
| --- |
| USE NutrientsDB;  CREATE PARTITION FUNCTION NutrientsDB\_PartitionRange (int)  AS RANGE LEFT FOR VALUES (50, 100);  CREATE PARTITION SCHEME NutrientsDB\_PartitionScheme  AS PARTITION NutrientsDB\_PartitionRange  TO ([PRIMARY], NutrientsDB\_Part2, NutrientsDB\_Part3);  CREATE TABLE Cheese (cheeseID INT NOT NULL PRIMARY KEY,  Type varchar(255),  Calories INT NOT NULL,  Proteins INT NOT NULL,  Carbohidrates INT NOT NULL,  Fat INT NOT NULL,)  ON NutrientsDB\_PartitionScheme (cheeseID); |

1. Comando T-SQL para criação de um índice sobre a nova coluna CaloriesInCal, que corresponde aos valores da coluna Calories mas em calorias/100g. Este índice está armazenado fisicamente no grupo de ficheiros primário e é NON-CLUSTERED, uma vez que a ordem dos registos no índice não corresponde à ordem dos registos na tabela.

|  |
| --- |
| USE NutrientsDB;  ALTER TABLE Cheese ADD CaloriesInCal INT NOT NULL;  GO  UPDATE Cheese SET CaloriesInCal = (Calories/1000) Where CaloriesInCal = 0  GO  CREATE INDEX calories\_index  ON Cheese(CaloriesInCal)  INCLUDE (Proteins, Fat)  ON [PRIMARY]; |

**2. B+Tree index structures**

Para facilitar a resolução do exercício, atribuiu-se a cada valor fornecido no enunciado um nome mais curto. Estes novos nomes serão então os utilizados no desenho das árvores B+. Assim:

|  |  |
| --- | --- |
| **Parmesão** | PA |
| **Ilha** | IL |
| **Camembert** | CA |
| **Fresco** | FR |
| **Requeijão** | RE |
| **Azeitão** | AZ |
| **Alverca** | ALV |
| **Serra** | SER |
| **Alcobaça** | ALC |
| **Roquefort** | RO |
| **Flamengo** | FL |
| **Emmental** | EM |
| **Évora** | EV |
| **Creme** | CR |
| **Serpa** | SEP |
| **Quark** | QU |

Tabela 1 - Atribuição de novos nomes para facilitar a resolução do exercício

Como se trata de uma árvore que consegue conter até 3 valores, sabendo que m = 4, tem-se que o número de valores num nó folha pode ser:

até

Ou seja, um nó folha poderá ter 2 ou 3 valores a qualquer altura, na árvore B+.

Assim, como indicado no enunciado, realizaram-se as seguintes operações:

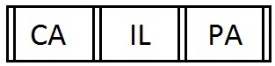
1. Inserção de Parmesão:

C:\Users\Asus\Dropbox\AOBD\Mini-Projectos\MP 1\Questão 2\1. insere parmesão.jpeg

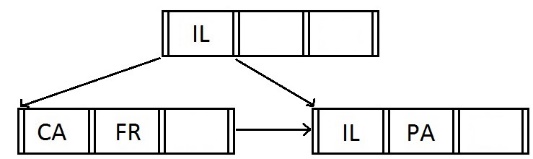
1. Inserção de Ilha:

C:\Users\Asus\Dropbox\AOBD\Mini-Projectos\MP 1\Questão 2\2. insere ilha.jpeg

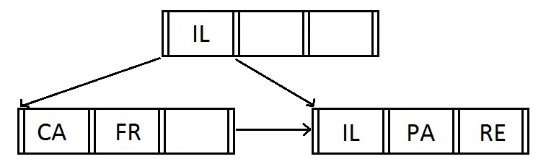
1. Inserção de Camembert:



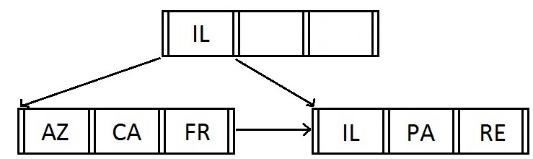
1. Inserção de Fresco:



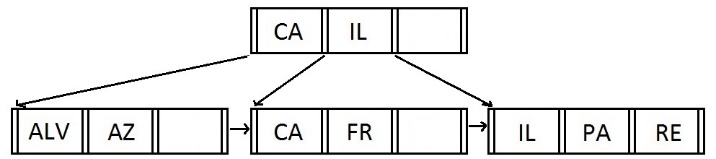
1. Inserção de Requeijão:



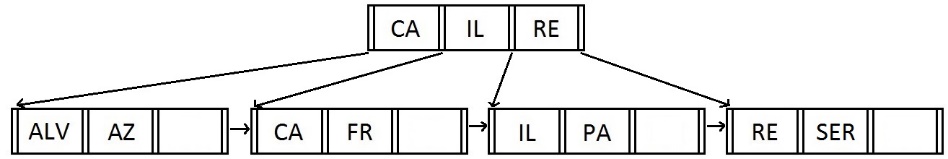
1. Inserção de Azeitão:



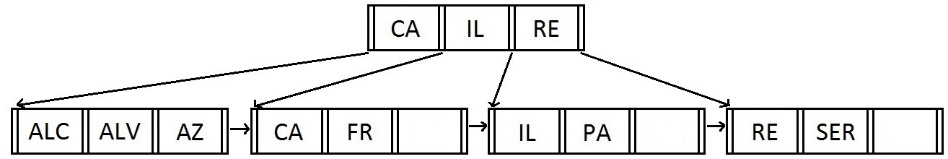
1. Inserção de Alverca:



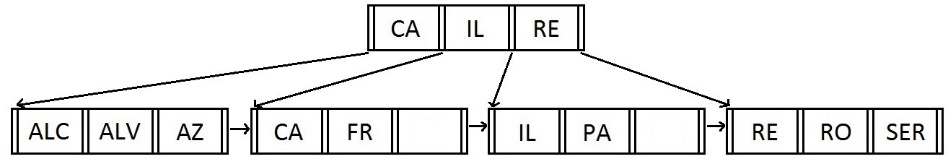
1. Inserção de Serra:



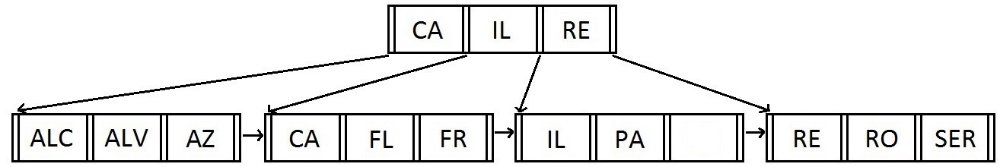
1. Inserção de Alcobaça:



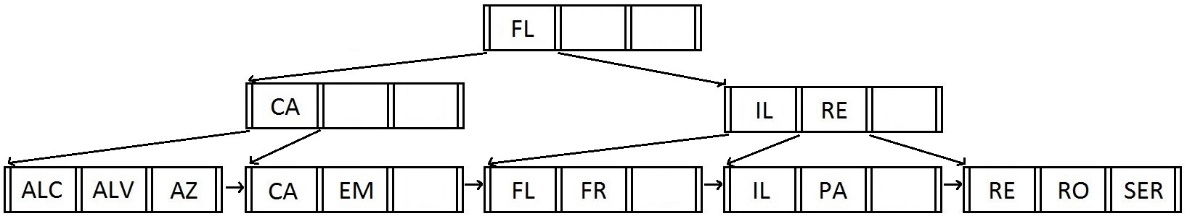
1. Inserção de Roquefort:



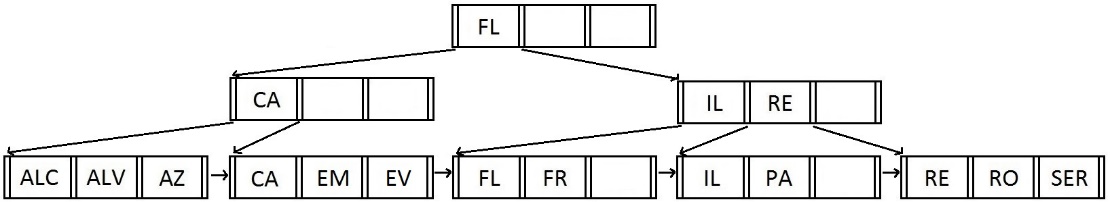
1. Inserção de Flamengo:



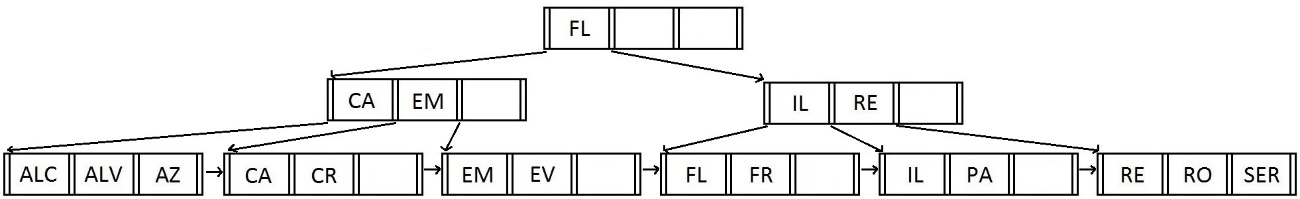
1. Inserção de Emmental:



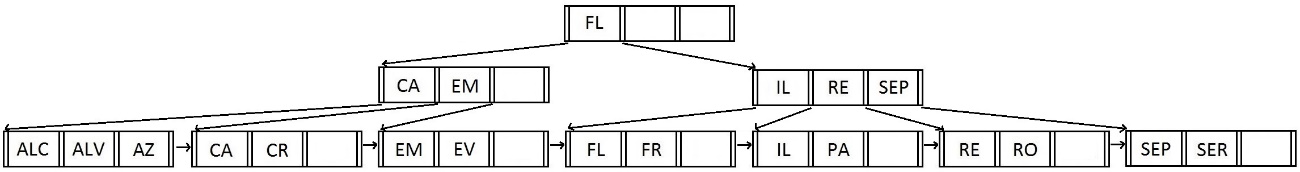
1. Inserção de Évora:



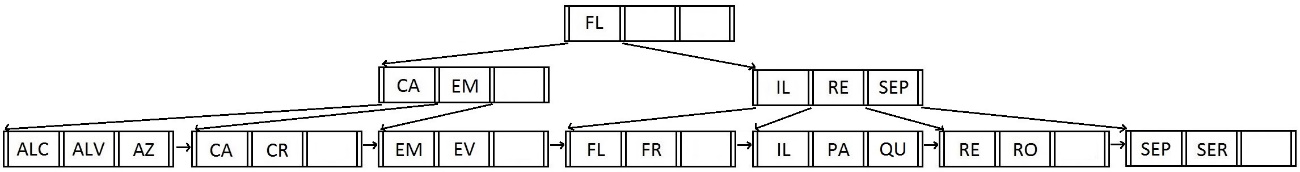
1. Inserção de Creme:



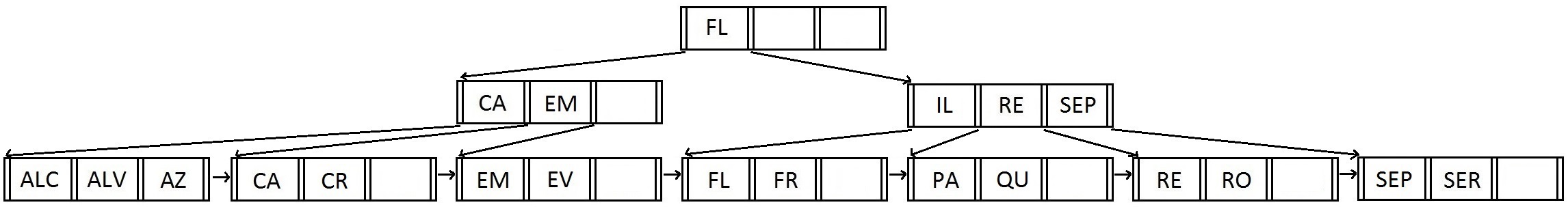
1. Inserção de Serpa:



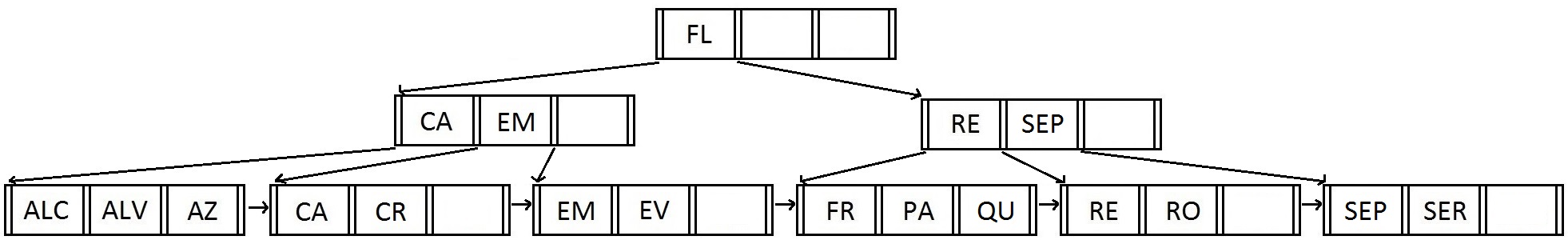
1. Inserção de Quark:



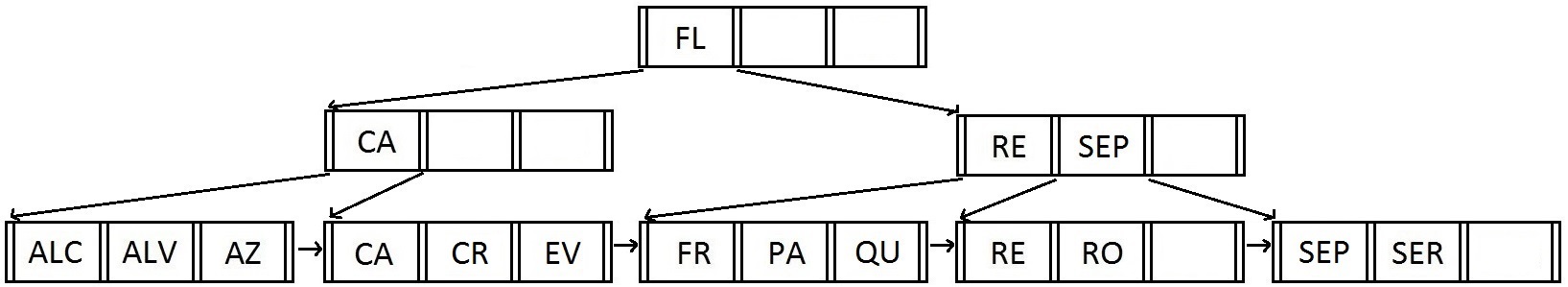
1. Remoção de Ilha:



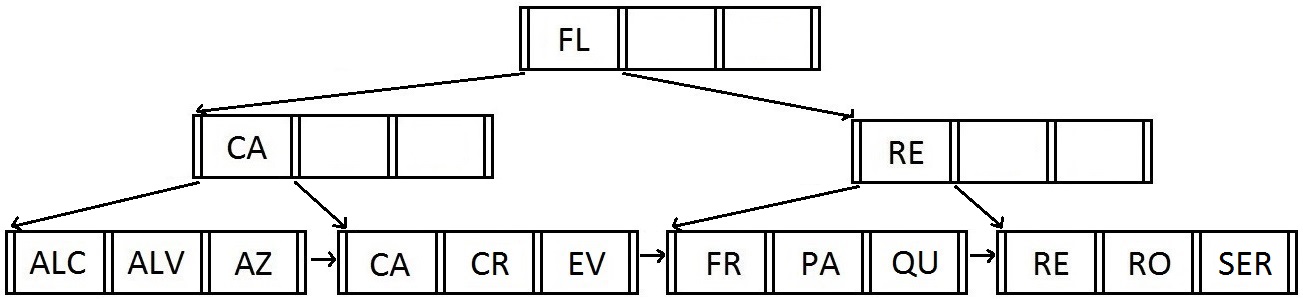
1. Remoção de Flamengo:



1. Remoção de Emmental:



1. Remoção de Serpa:



**3. Extendable hashing index schemes**

Para a realização deste exercício, teve-se em conta a seguinte representação binária das chaves fornecida no enunciado:

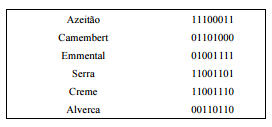
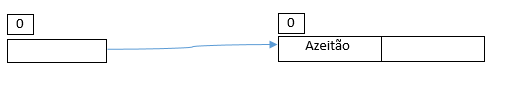


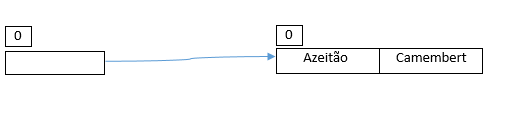
Tabela - Representação binária das chaves

Desta forma, foram inseridos os seguintes registos:

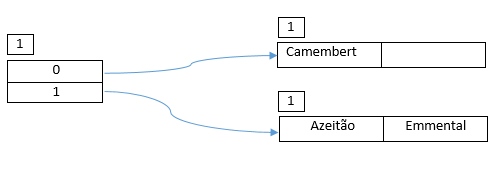
1. Inserção do registo Azeitão (11100011):



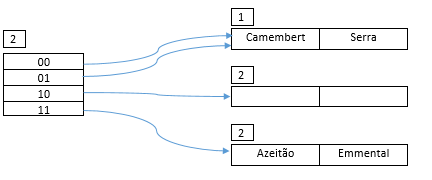
1. Inserção do 1º registo Camembert (01101000):

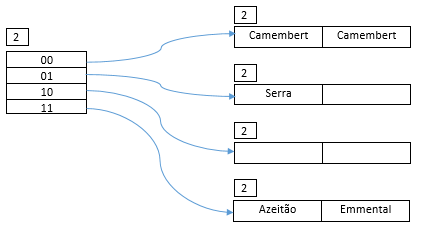


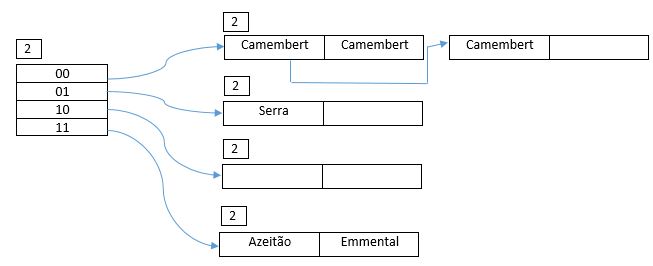
1. Inserção do registo Emmental (01001111):



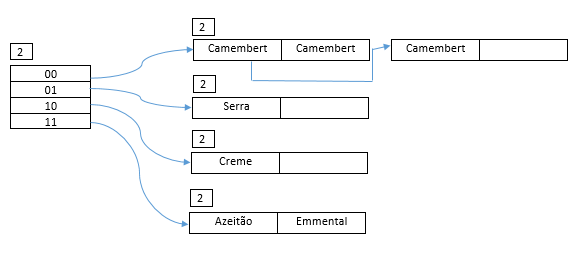
1. Inserção do registo Serra (11001101):



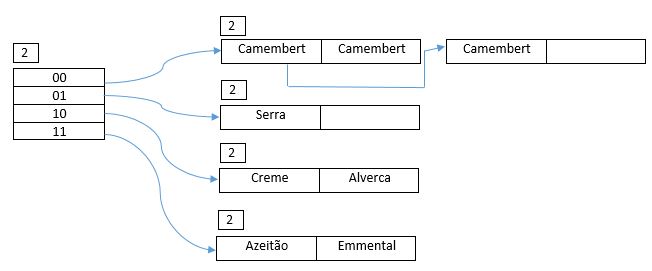
1. Inserção do 2º registo Camembert (01101000):  
   
2. Inserção do 3º registo Camembert (01101000):



1. Inserção do registo Creme (11001110):



1. Inserção do registo Alverca Curado (00110110):



**4. Estimating the cost of relational algebra operations**

1. Sendo **hi** a altura do índice em causa, **b** o número de blocos que contêm registos de *matching records* e **n** o número de registos que são *fetched*:

* Se o índice sobre climate-type (não-chave) é um índice non-clustered, então o custo será:

Número de I/Os = hi \* (tT + tS) + n \* (tT + tS)

Mas como o tempo de procura e transferência não é conhecido então esses tempos são desprezados, resultando em:

Número de I/Os = hi + n

* Se o índice sobre climate-type (não-chave) é um índice clustered, então o custo será:

Número de I/Os =*hi \**(*tT* +*tS*)+*tS* +*tT* \*b

Mas como o tempo de procura e transferência não é conhecido, então esses tempos são desprezados, resultando em:

Número de I/Os = hi + b

* Estimativa do **pior caso**, em que cada bloco da relação *Location* é lido uma vez para cada bloco na relação *cheeseProvenance*:

bcheeseProvenance \* bLocation + bcheeseProvenance transferências de blocos

1000 \* 2300 + 1000 transferências de blocos =

2 301 000 transferências de blocos

* Estimativa do **melhor caso**, em que a relação *Location* cabe toda em memória:

bcheeseProvenance + bLocation transferências de blocos =

1000 + 2300 transferências de blocos =

3300 transferências de blocos

c) Número de I/Os = bcheeseProvenance + bLocation transferências de blocos +

custo de ordenar a relação *cheeseProvenance*

» No cálculo da ordenação da relação *cheeseProvencance* é utilizado o algoritmo **External Sort-Merge** que especifica:

* Número de transferências de blocos:

bcheeseProvenance \* ( 2 =

= 1000 \* ( 2 =

= 2000 \* 9 + 1000 =

= 19 000 transferências de blocos

» Para o cálculo do custo do algoritmo **Sort-Merge Join:**

1000 + 2300 + 19 000 = 22 300 transferências de blocos

**5. Query optimization and estimation of Join sizes**

De forma a estimar o número de tuplos resultantes da expressão indicada, estimou-se primeiro o número de tuplos que resultam da seguinte seleção:

Como a expressão apresentada é do tipo , então o número de tuplos resultantes é:

( uma vez que corresponde à multiplicação do número de páginas de *Location* pelo número de tuplos em cada página, ou seja, 2300 x 100 = 230000)

Após ser estimado o número de tuplos da seleção, estimou-se o número de tuplos da seguinte expressão:

Como a interseção das duas relações () corresponde à coluna *region-name*, que é *key* da relação resultante da seleção indicada anteriormente, sabe-se que:

Logo, conclui-se que o número de tuplos resultantes da expressão nunca será maior do que o número de tuplos em S: 120000 (uma vez que corresponde à multiplicação do número de páginas de *CheeseProvenance* pelo número de tuplos em cada página, ou seja, 1000 x 120 = 120000).

Admitindo que *region-name* é *foreign key* de *CheeseProvenance*, pode-se afirmar que o número de tuplos resultantes da expressão é exatamente o número de tuplos em S, ou seja, 120000.

Nota: Após ser calculado o número de tuplos resultantes da expressão, conclui-se que o valor estimado do número de tuplos da seleção é irrelevante, logo poderia não ter sido calculado neste exercício.

**6. External talk: Casos reais na administração de Bases de Dados**

1. Tendo como objetivo a grande disponibilidade das bases de dados geridas pelo DBA, são realizadas réplicas das mesmas em diferentes servidores. Assim, para garantir a existência de um backup que possa ser utilizado caso ocorram falhas na base de dados principal, por exemplo uma falha de energia ou mesmo inoperabilidade total, é replicado o conteúdo da base de dados principal por todos esses servidores. No entanto, ao utilizar esta arquitetura, surge um trade-off: investimento em hardware que não contribui ativamente para a funcionalidade pretendida, gerando um custo que não é aproveitado e rentabilizado. Uma das técnicas usadas e referidas pelo engenheiro Wilson Lucas para minimizar o impacto deste trade-off consiste na utilização do servidor onde se encontra replicada a base de dados principal (em alguns casos com hardware de melhor qualidade do que o servidor principal) para o processamento de dados, com o intuito de otimizar os acessos à base de dados e rentabilizar todo o investimento efetuado.
2. Sabendo que uma base de dados usada regularmente por uma organização está associada a uma constante atualização de dados da mesma, é necessário que a mesma esteja permanentemente a ser otimizada. Para isso, a monitorização dos logs da base de dados tem um papel fundamental, uma vez que permite uma recolha de informação útil relativamente às operações realizadas/conteúdo acedido. Por exemplo, caso aumente o número de ocorrências de uma determinada query, será útil criar um índice com o intuito de otimizar a mesma. Assim, através desta monitorização, é possível garantir a constante otimização da base de dados.