

**Database Administration and Tuning**

## Mini-Project 1 - Solution

**Question 1 – SQL Server Databases**

Present SQL Server T-SQL commands for accomplishing the following tasks:

- (a) Create a database named *NutrientsDB*, containing **one log file** and **three different data files**, in **three distinct filegroups** (i.e., one data file in each filegroup). The log file should have an initial size of 25MB and a maximum size of 250MB. All data files should have an unlimited maximum size, except the one in the primary filegroup, which should have a maximum size of 1GB). The first data file on the first secondary filegroup should have an initial size of 100MB, and the remaining files should have an initial size of 50MB. All files should grow at a rate of 50%, except for the data file in the primary filegroup, which should grow by 5MB, every time this is required.

**Solution:**

```
create database NutrientsDB on
primary (
    name = 'ndb1',
    filename = 'c:\Users\Public\Documents\ndb1.mdf',
    size = 50MB,
    maxsize = 1GB,
    filegrowth = 5MB
),
filegroup ndb2 (
    name = 'ndb2',
    filename = 'c:\Users\Public\Documents\ndb2.ndf',
    size = 100MB,
    maxsize = unlimited,
    filegrowth = 50%
),
filegroup ndb3 (
    name = 'ndb3',
    filename = 'c:\Users\Public\Documents\ndb3.ndf',
    size = 50MB,
    maxsize = unlimited,
    filegrowth = 50%
),
log on (
    name = 'ndblog',
    filename = 'c:\Users\Public\Documents\ndblog.ldf',
    size = 25MB,
    maxsize = 250MB,
    filegrowth = 50%
);
```

- (b) Create a table named **Cheese** in the **NutrientsDB** database. The table should have a numeric attribute named **cheeseID**, that identifies the individual records, an alphanumeric attribute named **Type**, and four other numeric attributes named **Calories**, **Proteins**, **Carbohydrates**, and **Fat**. The table should be partitioned so that all tuples where *cheeseID* is less or equal than 50 are physically stored in the primary filegroup, all tuples where the *cheeseID* is greater than 50, but less or equal than 100, are physically stored in the first secondary filegroup, and the remaining tuples are physically stored in the second secondary filegroup.

**Solution:**

```
create partition function ndbrange (int)
as range left for values (50,100);

create partition scheme ndbpartscheme
as partition ndbparts to ([primary], ndb2, ndb3);

create table cheese (
    cheeseID int,
    type varchar(100),
    calories int,
    proteins int,
    carbs int,
    fat int,
    primary key (cheeseID))
on ndbpartscheme(cheeseID);
```

- (c) In the table named Cheese, the amount of calories is stored in an attribute named **Calories**, in Kcals per 100 grams. Create an index over the table with a search key corresponding to the calories in cals per 100 grams, including the amount of protein and fat as additional attributes that are not part of the search key. The index should be physically stored in the primary filegroup. Indicate also if the index is clustered or non-clustered, justifying.

**Solution:**

```
alter table cheese
add calgrams as (calories * 1000);

create index calprotfat
on cheese(calgrams)
include (proteins, fat)
on [primary];
```

O índice será secundário porque já existe um índice primário na tabela (para a chave primária da tabela).

<b>Question 2 – B+Tree Index Structures</b>
---

Consider the problem of inserting the following keys, in the given order, into an empty B+-tree

where nodes can hold up to 3 values:

*Parmesão, Ilha, Camembert, Fresco, Requeijão, Azeitão, Alverca, Serra, Alcobaça, Roquefort, Flamengo, Emmental, Évora, Creme, Serpa, Quark*

- (a) Draw the tree after each insertion.
- (b) Delete the following keys from the B+tree data structure from the previous exercise: *Ilha; Flamengo; Emmental; Serpa*. Draw the tree after each deletion.

**Solution:** see other file.

### 3 – Extendable Hashing Index Schemes

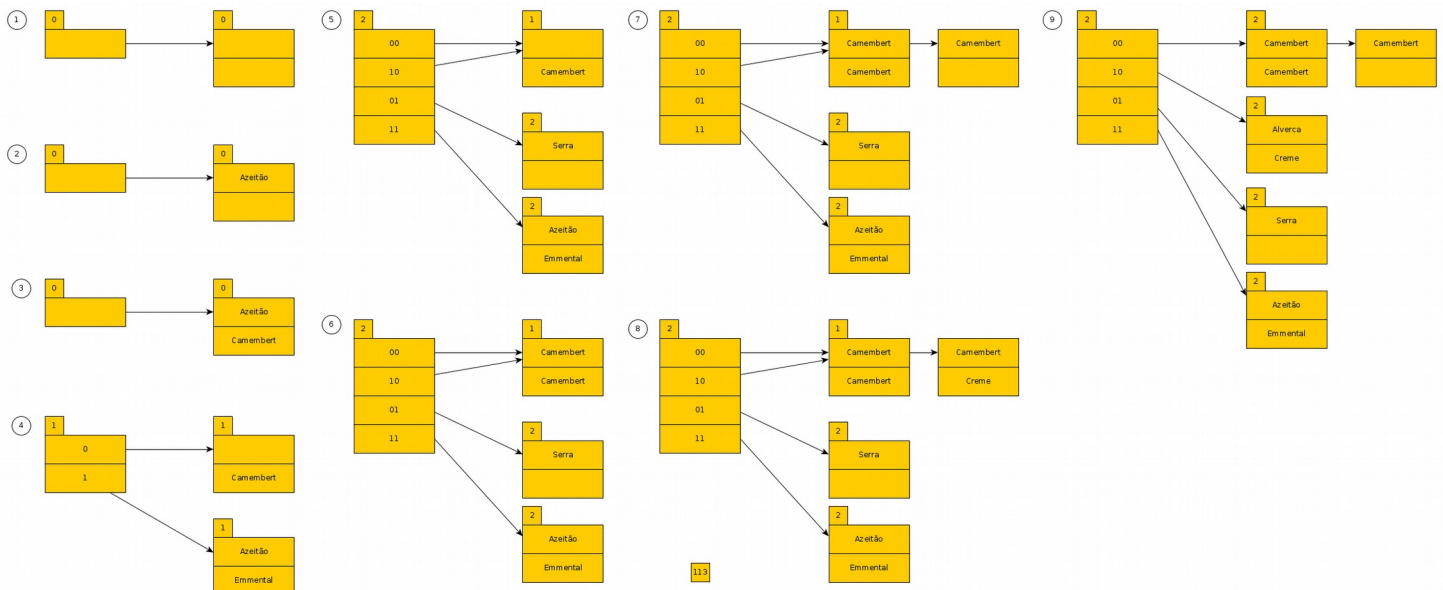
Consider the extendable hashing indexing mechanism introduced in the theoretical classes. Show how data records with the following keys can be stored in buckets which individually can hold 2 records, using extendable hashing and considering the **least significant bits first** in the directory of the resulting data structure.

*Azeitão, Camembert, Emmental, Serra, Camembert, Camembert, Creme, Alverca curado*

Consider that binary representations for keys were determined by a simple hash function, resulting in:

Azeitão	11100011
Camembert	01101000
Emmental	01001111
Serra	11001101
Creme	11001110
Alverca	00110110

**Solution:**



#### 4 – Estimating the Cost of Relational Algebra Operations

Consider the following relational schema:

*CheeseProvenance*(cheese-name, region-name)  
*Location*(region-name, climate-type)

The relation *CheeseProvenance* stores information about the region where each cheese type is produced, and the relation *Location* stores information about the the regions that produce cheese.

All tuples have fixed size. The relation *CheeseProvenance* takes 1000 blocks and the relation *Location* has 2300 blocks. Each page of *CheeseProvenance* contains 120 tuples and each page of *Location* contains 100 tuples.

Compute the number of I/Os performed by each of the following algorithms:

- Selection on the *Location* relation where the filtering condition is *climate-type* = 'Dry', assuming there is an index on the table over the attribute *climate-type*.
- Block Nested Loop Join, with *CheeseProvenance* as the outer relation and the join condition is on *region-name*. Present the costs of the worst and best cases.
- Sort-Merge Join, assuming that only the relation *Location* is ordered on *region-name*, the relation *CheeseProvenance* is ordered on *cheese-name* and that you can have 3 pages in memory when sorting the relations.

**Solution:**

- If we assume it is a clustered index, then cost =  $h_i + b$ , where  $h_i$  is the tree height and  $b$  is the number of blocks of *Location* that contain tuples satisfying the condition *climate* = dry.

If we assume it is a non-clustered index, then  $\text{cost} = H_i + n$ , where  $n$  is the number of Location tuples satisfying the condition.

(b) BNL  $\text{cost} = bR * bS + bR$  (worst case)  $\Rightarrow \text{cost} = 1000 * 2300 + 1000 = 2301000$  IOs  
 $\text{cost} = bR + bS$  (best case)

(c) SMJ  $\text{cost} = bR + bS + \text{cost}(\text{sorting CheeseProvenance})$   
 $\text{cost}(\text{sorting CheeseProvenance}) = bCP * (2 * \text{ceiling}(\log_2(bCP/3)) + 1) =$   
 $1000 * (2 * \text{ceiling}(\log_2(1000/3)) + 1) = 19000$  IOs  
 $\text{cost} = 1000 + 2300 + 19000 = 22300$  IOs

## 5 – Query Optimization and Estimation of Join Sizes

Consider the two relations of Question 4. Consider also the following information, regarding the two relations:

$$V(\text{climate}, \text{Location}) = 20$$

Estimate the number of tuples that results from the expression:

$$\text{CheeseProvenance} \bowtie (\sigma_{\text{climate}='dry'} \text{Location})$$

**Solution:**

Result size of  $(\sigma_{\text{climate}='dry'} \text{Location}) = \text{Nb tuples Location} / V(\text{climate}, \text{Location}) = 11500$

Result size of  $\text{CheeseProvenance} \bowtie (\sigma_{\text{climate}='dry'} \text{Location}) = \text{Nb tuples of CheeseProvenance} = 120 * 1000 = 120,000$ ,

because the intersection of schemas of Location and CheeseProvenance is region-name, region-name is the key of Location (and also of  $\sigma_{\text{climate}='dry'} \text{Location}$ ) and is the foreign key of CheeseProvenance referencing Location.

## 6 - External talk: Casos Reais na Administração de Bases de Dados

Answer the following questions, on the subject of the invited talk given by engineer Wilson Lucas, that you had the opportunity to attend on the 24th of May 2015.

(a) One of the tasks of a DBA is to assure the high availability of the Databases being managed. This can be done in several ways. However, independently of the technique used, there are always trade-offs that one must take into account when choosing how to implement it (or even if it is worth implementing). Name and explain one such trade-off.

**Solution:**

To assure full availability, one usually needs to replicate the data. This can be done at different levels, ranging from replicating the database, to replicate the full datacenter. The major trade-off here is the cost of having redundant hardware, software, and even people ready at all times.

(b) In theory, once our database is fully optimized, it should not be necessary to change it any further. In practice, on a database that is being used in a functioning organization, this is not the case. Explain why.

**Solution:**

In reality, an organization is constantly changing, according to the needs of its employees, its clients, the markets, and so on. Thus, its databases are also constantly changing, and every change may need new optimizations.