Exercise Sheet 2

Exercise 1 (Round-robin partitioning)

Round-robin partitioning is a variant of horizontal partitioning that assigns tuples turn by turn to each partition.

1. What benefits/limitations do you expect from that partitioning scheme?
2. There is no explicit instruction to obtain round-robin partitioning in Oracle. How would you approximately simulate round-robin partitioning in Oracle?

Exercise 2 (choosing partitioning strategy)

Some DW must partition a world population table on 'birthday' column. What partitioning strategy should you adopt if all queries are of the following form, the only varying parameter being the country? We assume a non-parallel setting where partitions can not be scanned in parallel.

```
SELECT first_name, last_name
FROM population
WHERE birthday>01.01.1990
  AND birthday<31.12.2000
  AND country='France';
```

Exercise 3

A DWH records sales events according Geography (country, city, and region within country) and per product category. Which of the following attribute-clustering schemes seems the most promising (justify):

1. attribute-clustering with respect to the 2 columns: (country,city,region) and category
2. attribute-clustering with respect to the 2 columns: (city,region,country) and category
3. attribute-clustering with respect to the 2 columns: (country,region,city) and category
4. attribute-clustering with respect to the 4 columns: country, city, region and category
5. attribute-clustering with respect to the 4 columns: category, city, region, country
6. attribute-clustering with respect to the 4 columns: region, city, country and category

Exercise 4 (Indexes, views)

The 2 tables below represent parts of a database recording the inventory of a music store. PK indicates a primary key, and → a foreign key. For questions 1 and 2 you will assume that there are no additional records beyond those displayed on the figure whereas for questions 3, 4 and 5 the figures are a sample of existing records. Note that some records might include `NULLs`.

Inventory

Work_id (PK)	Price	C→ Composer.Id	S→ Style.Id	ISBN	Binding
1	24.0	1	1	1	Case
3	12.4	4	2	2	Comb
2	23.0	1	1	1	Comb
4	24.0	4	2	2	Comb
5	10.0	4	1	3	Crisscross
6	39.99	1	1	4	Stitched
7	20.0	1	1	4	Comb
9	30.0	2	2	6	Case
10	8.99	2	2	7	Case
8	59.99	4	1	5	Case

Style

Id (PK)	Name
1	Vocal
2	Piano

Composer

Id (PK)	Name
1	Rameau
2	Glück
3	Sibelius
4	Dvořák

1. Represent a bitmap Join index that records the composer of each work in the inventory.

   .....

2. Apply the run-length encoding viewed in the lecture to the bitvector that begins with a "1".

   .....

3. For each of the following kind of queries, where the only parameters that may vary are the ones **underlined**, indicate which index or combination of indexes should be built (i.e., which index helps best optimize the query at a reasonable cost - we are assuming that indexes speedup reads even on a table with only thousands of records). You will assume the following cardinality for each attribute: `Work_id` has 10^7 distinct values, `Composer.name` has 10^3, `Styles.name` has 20, `Price` has 10^4, `Binding` has 10, and `ISBN`: 3×10^6:

(a) Q1:

`SELECT COUNT(*) FROM Inventory WHERE Price > x And Price < y ;`

(b) Q2:

```
SELECT SUM(Price) FROM Inventory I, Style St WHERE I.S=St.Id AND (Name = x OR St.Name = y) ;
```

(c) Q3:

```
SELECT Name FROM Composer WHERE Id=x ;
```

(d) Q4:

```
SELECT Work_id FROM Inventory WHERE S=x AND Binding<>y ;
```

(e) Q5 (Bonus). *[Think carefully. N.B.:* `UPPER` *converts a string to uppercase]*:

```
SELECT MIN(Price) FROM Inventory WHERE ISBN=x AND UPPER(Binding)=y ;
```

4. Give the SQL instruction to create a materialized view `MyView(Composer, Binding, Nb, Worth)` using at least one `ROLLUP` statement [1]. The view should record, for the two groups `(Composer, Binding)` and `(Composer)`, the number of works in the group and the total worth of the group (cf picture).

```
> SELECT * FROM MyView;

Composer Binding Nb  Worth
-------- ------- -- ------
Glück    Glück    2  38.99
...
Glück             2  38.99
...
Rameau           75 150.33
...
```

5. Could `MyView` be exploited to rewrite query below? If yes, give the rewriting. If not, explain what should be added to the view to allow the rewriting then give the rewriting.

```
SELECT Composer, AVG(Price)
FROM Inventory
GROUP BY Composer
```

Exercise 5 (Partitioning in MongoDB)

We record sales as JSON objects

```
db.sales.insert({client_id: "a000", product_id: 300001, date_vente: new Date('Jun 07, 2014'), ...})
```

When inserting those data in the database, we assume there are 100GB of sales: 40GB in France, 20GB in both Germany and Belgium, 5GB in each of the remaining 4 following countries UK, Italy, Spain, Switzerland. Sales are distributed roughly uniformly among dates and places. We assume we have a cluster of 1000 machines and use MongoDB's default partitioning options.

1. If we partition on the country, what will be the repartition of shards?
2. if we partition on a 100 machine cluster, on $\{country, sales\_date\}$?

[1] If you don't manage to use ROLLUP you can use `GROUPING SETS` instead but I will remove .5 point