# Distributed databases exercises

DBM1

October 13, 2015

## Exercise 1

Consider a database table tracking performance stats of web requests. The schema is:

*Requests*(request_id, url, timestamp, client_ip, web_server_ip, request_size, response_size, http_code, latency_ms)

The primary key is "request_id". HTTP code is the return code from the server like 200 (if everything went fine), 404 (page not found), 500 (server error), etc.

1. Imagine that by far the most common query against this table is:

   ```
   SELECT url, AVG(latency_ms)
   FROM Requests
   GROUP BY url;
   ```

   Fragment the requests table into two tables, using vertical fragmentation, in order to provide the best performance for this query. Express your fragments as a projection operation over the original table.

2. For each of the following horizontal fragmentations, state whether it is complete, disjoint, neither or both. For strings comparison, we consider the lexicographical order.
   a) $Requests_1 = \sigma_{url<"http://www.google.com"}(Requests)$,
      $Requests_2 = \sigma_{url>="http://www.apple.com"}(Requests)$
   b) $Requests_1 = \sigma_{request\_size<response\_size}(Requests)$,
      $Requests_2 = \sigma_{request\_size\geq response\_size}(Requests)$
   c) $Requests_1 = \sigma_{http\_code=200}(Requests)$,
      $Requests_2 = \sigma_{http\_code=404}(Requests)$
   d) $Requests_1 = \sigma_{latency\_ms\geq 0}(Requests)$,
      $Requests_2 = \sigma_{latency\_ms<0}(Requests)$

## Exercise 2

Consider the following tables, each being stored on a different node:

1. *Player*(player_id, first_name, last_name, email_address, subscription_plan)
2. *MatchPlayed*(match_id, date, home_player_id, visitor_player_id, game_name, home_score, visitor_score, log)
3. *Games*(game_name, description)
4. *GamePricing*(game_name, subscription_plan, fee_per_game)

Primary keys are the underlined attributes. The following relationships between attributes exist:

- *MatchPlayed*(home_player_id) → *Player*(player_id)
- *MatchPlayed*(visitor_player_id) → *Player*(player_id)
- *MatchPlayed*(game_name) → *Games*(game_name)
- *GamePricing*(game_name) → *Games*(game_name)
- *Player*(subscription_plan) → *GamePricing*(subscription_plan)

Imagine we have the following statistics about the number of bytes needed to represent the content of the tables (TB is a terabyte, i.e., $10^{12}$ bytes):

| Table | Total size |
|---|---|
| Player | 0.01 TB |
| MatchPlayed | 50 TB |
| Games | 0.001 TB |
| GamePricing | 0.001 TB |
| Games ⋈ GamePricing | 0.002 TB |
| Games ⋈ MatchPlayed | 70 TB |
| MatchPlayed ⋈ Player | 110 TB |
| Player ⋈ GamePricing | 0.011 TB |

We consider as a cost metric the amount of data that has to be transferred between sites to evaluate query plans. We want to compute the following query:

```
SELECT first_name, last_name, Games.game_name, description, SUM(fee_per_game)
FROM Player
JOIN MatchPlayed ON player_id = home_player_id
  OR player_id = visitor_player_id
JOIN Games ON MatchPlayed.game_name = Games.game_name
JOIN GamePricing ON Games.game_name = GamePricing.game_name
  AND Players.subscription_plan = GamePricing.subscription_plan
GROUP BY Player.first_name, Player.last_name, Games.game_name, description;
```

1. Formulate in english what this query does.
2. Compute the cost of the following plan:
   1. Send *Games* to node 2.
   2. Compute $Games \bowtie MatchPlayed$ on node 2, and send result to node 1.
   3. Send *Player* to node 4.
   4. Compute $Player \bowtie GamePricing$ on node 4, and send result to site 1.
   5. Compute the aggregate on node 1.
3. Assume we are considering plans that send all base tables to a single node and compute the full query on it. Which node is better to have the final results collected on? Why?
4. Imagine that we fragment *MatchPlayed* by match_id into 15 fragments. Now want to sort *MatchPlayed* by "date" (and that this is not necessarily the same order than match_id). The final sorted relation can be fragmented across 10 nodes. Briefly describe an efficient algorithm for sorting the data, describing how data flows between nodes.