EXERCISE 10

Andrea Crotti (#299466), Can Liu (#286105), Sebastian Roidl (#281941)

13 Gennaio 2010

Exercise 10.1[Query optimization]:

1.

```
SELECT gift.name, supplier.name, COUNT(*)
FROM gift, supplier, shipment
WHERE shipment.gift = gift.gift_id AND shipment.supplier = supplier.supplier_id
AND gift.price > 190 AND supplier.place IN
(SELECT DISTINCT place.place_id
FROM employee, place
WHERE employee.place = place.place_id AND employee.nationality = 'German' )
GROUP BY gift.name, supplier.name
```

a) The selection (IN expression) on supplier.place

We know that there are 50 different nationalities, assuming that they are uniformly distributed we get:
With the line *SELECT DISTINCT place.place_ id* we get a selectivity of 1/1000, which are the number of possible disinct places.

But we also have a filter on the nationality, assuming the cities are equally distributed we have to divide the places by the number of nationalities. $selectivity = 1/(1000/50) = 1/20$

b) The selection on gift.price

Prices are equally distributed in the range (11, 210).
Choosing gift.price $>$ 190 we get
$S = (210 - 190)/(210 - 11) \approx 1/10$

c) The join on shipment.gift and gift.gift_id

The general formula for join is $1/max(V(S,y), V(R,y))$
So in this case 1/6000 (number of distinct possible gifts).

d) The join on shipment.supplier and supplier.supplier_id

$selectivity = 1/500$

e) The join on employee.place and place.place_id

$selectivity = 1/1000$ (the number of employees)

2. Sketch a query plan for this query that would result in the minimum amount of work given no indices and the selectivity estimates you gave above. Be sure to indicate what join algorithm you would use for each join. Assume you have sufficient memory to fit all relations and any intermediate data structures in memory.

For all the joins, provided we have enough memory space, we use Block Nested Loop join algorithm.
We could load as many pages of the bigger relation as can be fit in the available memory, and load all such tuples into a hash table, then repeatedly scans the smaller relation.
It is the cheapest way when we have enough large input buffer.

3. Reccommend a set of indices that will improve the performance of this query at most

In general we should create a clustered hash index for equalities and a b+tree index for attributes involved in inequalities.

- clustered hash index on place.place_id
- clustered hash index on gift.gift_id
- clustered b+tree index on gift.price
- clustered hash index on employee.nationality