Ferhat Elmas 214805

Advanced Databases Hw 1

1)

1. SELECT * FROM Actor A WHERE A.net_worth > (SELECT A1.net_worth FROM Actor A1 WHERE A1.name='Will Smith')
2. SELECT * FROM Actor EXCEPT (SELECT A.name, A.net_worth, A.hometown FROM Actor A, Cast C WHERE A.name=C.actor AND C.movie='Raising Arizona')
3. SELECT * FROM Actor A WHERE A.net_worth > (SELECT MAX(net_worth) FROM Actor WHERE hometown='New York')

2)

1. πA1.name, A1.net_worth, A1.hometown(σA1.net_worth>A2.net_worth ∧ A2.name='Will Smith'(ρA1(Actor)⋈ρA2(Actor)))
2. Actor-πActor.name, Actor.net_worth, Actor.hometown(σActor.name = Cast.actor(Actor ⋈σmovie='Raising Arizona'(Cast)))

3)

1. SELECT S.id FROM Suppliers S WHERE 100000000 <= (SELECT AVG(P.price) FROM Product P WHERE P.supplier=S.id)
2. SELECT S.id, S.name, P.price, P.name FROM Supplier S, Product P WHERE P.supplier=S.id AND P.price=(SELECT MAX(P1.price) FROM Product P1 WHERE P1.supplier=S.id)

Query Operators

1) *Simple Hash-Join*:

- Scans R repeatedly to partition R as much as possible to fit it in a hash table in memory.
- After each scan and build of hash table, S (larger) is scanned for tuples of R that are in the memory and a probe is made for a match.

GRACE Hash-Join:

- Partitions R and S into subsets where partitions of R are approximately equally divided. For each subset, there is a different buffer, each tuple is put its respective buffer according to hash function, when whole relations are scanned, buffers are written to disk.
- After partition phase, each subset corresponding to R and S is read into memory and a hash table is built from R and after hashing the tuple of the corresponding subset of S, it is probed for a match in hash table. If there is a match, it is in the result set, else continued with next tuple of the set of S.

Since Simple Hash-Join tries to put as much of R into memory and scans for S repetitively until R is fully scanned, if most of R fits into main memory, it perfons well because most of R and S are touched only once and only extra parts are written into disk and read again. However, when memory is small, there will be many passes and scans of R and S over and over.

On other hand, GRACE perform well in small memory by preventing repeatedly scanning R and S. In large memory, proportional to R, simple hash-join perform better because it touches relations once since R fits into memory while GRACE touches twice becuase it scans R and S for partitioning and then each subset is read again for probing which make totally two reads.

We can see this simple result in fig 7. In smaller memory, GRACE is better but while memory is increasing, we see that simple hash-join outperforms GRACE.

2) In hash based algorithms, tuples of S are used for probing or are copied into disk to be used later and remaining disks in the disk are processed sequentially. Therefore, they are in use one by one and never totally in memory. As a result, their partition doesn't matter at all. However, in sort-merge join algorithm, in phase 2, one block of memory is allocated and one run is read for both of R and S and concurrently merge is done in the all runs of R and S. Since we do sorting so that interleaved linear scans in runs encounter the sets at the same time, the size of both partitions are important.

Btw, I have blogged about the paper here, it is a summary of what I have learnt from the paper.