Implementation of Databases

Assignment 3

Participants:
(sorted in last name order)
Ulfet CETIN
Shreya KAR
Samuel ROY

Exercise 3.1 (Tableaux Containment and Minimization)

Given are the following tableaux:

T1

a1	a2	
b3	a2	(R)
a1	b4	(R)
5	b3	(R)
b4	5	(R)

T2

a1	a2	
b4	a2	(R)
b1	a2	(R)
a1	b3	(R)
b2	b4	(R)
b2	b1	(R)
b3	b2	(R)

T3

a1	a2	
b3	a2	(R)
a1	b2	(R)
b4	b1	(R)
b1	b2	(R)
b2	b3	(R)
b1	b3	(R)

For first question, we have used:

Aho, A. V., Sagiv, Y., & Ullman, J. D. (1979). Efficient optimization of a class of relational expressions. ACM Transactions on Database Systems, 4(4), 435-454. doi:10.1145/320107.320112

Just to provide you an easier access, here is the link:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.3427&rep=rep1&type=pdf

1. Find out if $T_i \subseteq T_j$ i.e., T_i T_j for i $\neq$ j, i, j $\in \{1, 2, 3\}$.

   We have gone through all possible mappings from $T_i \subseteq T_j$ i.e., T_i T_j for i $\neq$ j, i, j $\in \{1, 2, 3\}$.

   Yet, we only found that T1 $\subseteq$ T2 holds.

   Here is the mapping.

T2	to	T1
a1	$\rightarrow$	a1
a2	$\rightarrow$	a2
b1	$\rightarrow$	b3
b2	$\rightarrow$	5
b3	$\rightarrow$	b4
b4	$\rightarrow$	b3

   Here is mapped T2.

T2_mapped

a1	a2
b3	a2
b3	a2
a1	b4
5	b3
5	b3
b4	5

2. Write down the minimal tableau for T_i , i $\in$ 1, 2, 3.

T1

a1	a2	
b3	a2	(R)
a1	b4	(R)
5	b3	(R)
b4	5	(R)

T1 is already a minimal Tableau.

T2

a1	a2	
b1	a2	(R)
a1	b3	(R)
b2	b1	(R)
b3	b2	(R)

Mapping b4 $\rightarrow$ b1, this is the minimal Tableau

T3

a1	a2	
b3	a2	(R)
a1	b2	(R)
b4	b1	(R)
b1	b2	(R)
b2	b3	(R)
b1	b3	(R)

T3 is already a minimal Tableau.

Exercise 3.2 (Quant Graphs)

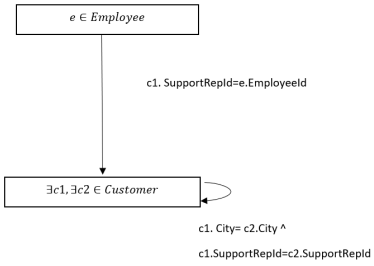
Given is the Chinook database schema (see Assignment 1).

1. Specify the following queries in TRC and draw the corresponding quant graphs, subsequently. Determine for each graph, if it contains a cycle. Explain in your own words what this means for the query.

   (a) EmployeeId and LastName of employees who have supported at least two different customers living in the same city.

   TRC

   $$\{< e.EmployeeId >, < e.LastName > | e \in Employee$$
   $$(\exists c1 \in Customer\ \exists c2 \in Customer$$
   $$(c1.City = c2.City \Lambda$$
   $$c1.SupportRepId = c2.SupportRepId \Lambda$$
   $$c1.SupportRepId = e.EmployeeId)$$
   $$)\}$$

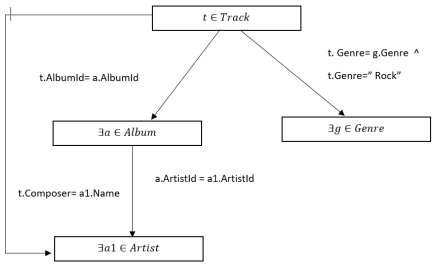
   

   This Query contains a cycle as there exists a predicate cycle from relation Customer to the node itself. The problem with a cyclic quant graph is that There is no sequence of semijoin operators that compute the correct result for arbitrary database states. However, it doesn't make a difference in particular query as the semijoin between the same relation (Customers) will yield the same result. In other words, the order here is trivial to compute the intermediate result as the same relation is involved.

   (b) Track name and composer of all tracks, which are from the genre "Rock" and where the artist is also the composer. Solution:

   TRC:

   $$\{< t.Name >, < t.Composer > | t \in Track$$
   $$(\exists a \in Album$$
   $$\exists a1 \in Artist$$
   $$\exists g \in Genre$$
   $$(t.AlbumId = a.AlbumId \Lambda$$
   $$a.ArtistId = a1.ArtistId \Lambda$$
   $$t.Genre = g.Genre \Lambda$$
   $$t.Composer = a1.Name \Lambda$$
   $$t.Genre = "Rock")$$
   $$)\}$$

   This query contains a cycle as there exists a predicate cycle. What it means is that there is no sequence of semijoin operators that compute the correct intermediate result for given relations Track,Album and Artist.It also implies that there are certain intermediate DB states where no possible sequence of operands can achieve a reduction of the involved range relations.
   i.e. Rel1 semijoin with Rel2 where $Rel1, Rel2 \in \{Track, Album, Artist\}$ will always yield Rel1 as result

Exercise 3.3 (Join Implementations)

Let relations R and S have the following properties:

- R has 10,000 tuples and has 10 tuples per page
- S has 2000 tuples and also has 10 tuples per page
- Both the relations are stored as simple heap files and neither relation has any indexes built on it.

The total number of buffers available is 52. Now considering the join R $\bowtie_{R.a=S.b}$ S assuming that attribute b on relation S is the primary key for S, answer the following questions: (The cost of writing out the result should be uniformly ignored.)

1. Calculate the I/O requirements of a page-oriented simple nested loop join. Solution:
   M=number of pages for relation R = 10000/10 =1000
   N= number of pages for relation S = 2000/10 =200
   I/0 costs of page oriented Simple nested loop join (taking R as outer relation)= M+M*N
   =1000+1000*200
   =2,01,000
   I/0 costs of page oriented Simple nested loop join(taking S as outer relation)= N+N*M
   =200+200*1000
   =2,00,200

2. Calculate the I/O requirements of a block nested loop join. Solution:
   B=52
   I/O requirements of Block Nested Loop join (taking R as outer relation)=M+[M/(B-2)]*N
   =1000+[1000/(52-2)]*200
   =1000+20*200
   =5000
   I/O requirements of Block Nested Loop join (taking S as outer relation)=N+[N/(B-2)]*M
   =200+[200/(52-2)]*1000
   =200+4*1000
   =4200

Exercise 3.4 (Cost Estimation)

1. Consider the following relations and the number of distinct values for the different attributes V (R, a). Each of the relations has a size of 1000 tuples. Determine an optimal order for joining all three relations. For each intermediate step determine the sizes of the results in terms of tuples. The cost of each plan is the sum of all intermediate result sizes. Select the best plan based on your results.

R(a,b)	S(b,c)	T(c,d)
V(R,a) = 100		
V(R,b) = 200	V(S,b)=100	
	V(S,c)=500	V(T,c)=20
		V(T,d)=50

The result size of joining relations R(X,Y) and S(Y,Z) can be approximated by:

$$T = \frac{T(R) * T(S)}{max(V(R,Y), V(S,Y))}$$

(R $\bowtie$ S) $\bowtie$ T , R $\bowtie$ (S $\bowtie$ T) are the possible plans:

R(a,b) with T(R)=1000 and V(R,b)=200
S(b,c) with T(S)=1000 and V(S,b)=100 and V(S,c)= 500
T(c,d) with T(T)=1000 and V(T,c)=20

Size and Cost Calculation for(R $\bowtie$ S) $\bowtie$ T:

$$\begin{aligned} T(R \bowtie S) &= \frac{T(R) * T(S)}{max(V(R,b), V(S,b))} \\ &= \frac{1000 * 1000}{200} \\ &= 5000 \end{aligned}$$

$$\begin{aligned} T(R \bowtie S) \bowtie T &= \frac{T(R \bowtie S) * T(T)}{max(V(R \bowtie S, c), V(T,c))} \\ &= \frac{5000 * 1000}{500} \\ &= 10000 \end{aligned}$$

Total Cost for(R $\bowtie$ S) $\bowtie$ T : 10000+5000=15000

Size and Cost Calculation for R $\bowtie$ (S $\bowtie$ T):

$$\begin{aligned} T(S \bowtie T) &= \frac{T(S) * T(T)}{max(V(S,c), V(T,c))} \\ &= \frac{1000 * 1000}{500} \\ &= 2000 \end{aligned}$$

$$T(S \bowtie T) \bowtie R = \frac{T(S \bowtie T) * T(R)}{max(V(S \bowtie T, b), V(T,b))}$$

$$= \frac{2000 * 1000}{200}$$
$$= 10000$$

Total Cost for(R $\bowtie$ S) $\bowtie$ T : 10000+2000=12000

Best plan is R $\bowtie$ (S $\bowtie$ T)

2. Consider the following relational schema and SQL query. The schema captures information about employees, departments, and finances (organized on a per department basis).

Suppliers(sid, sname, city)
Supply(sid, pid)
Parts(pid, pname, price)

```
SELECT P.pname
FROM Suppliers S , Parts P , Supply Y
WHERE S.sid=Y.sid AND Y.pid=P.pid AND
S.city= ' Aachen ' AND P.price <= 200 AND
Y.pid < 800
```

(a) Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations a decent query optimizer would choose (applying the heuristics selection before join, and projections are done as many and as early as possible).

$\rho(suppliersInAachen, \Pi_{sid}(\sigma_{city="Aachen"} Suppliers))$

$\rho(suppliedTo, \Pi_{pid}(suppliersInAachen \bowtie (\sigma_{pid<800} Supply)))$

$\Pi_{pname}(suppliedTo \bowtie (\Pi_{pid,pname}(\sigma_{price<=200} Parts)))$

(b) What indexes for single attributes might be of help in processing this query? Should it be clustered or unclustered indexes, using hash tables or B+ trees?

There are three where we do selection. The rest of the fields in WHERE clause, those will be handled with join operations.
{ Suppliers.city, Supply.pid, Part.price }

Field Name	Clustered or Unclustered	Hash Table or B+ Tree
Suppliers.city	Unclustered	Both hash table and B+ tree would work: 1) based on implementation, hash table may work bad if bucket approach is not used 2) B+ tree would work better, as records having their city field as "Aachen" will be placed consecutively if data records are stored (instead of data entries)
Supply.pid	Clustered	Both hash table and B+ tree would work: BUT B+ tree would work better most probably. This answer is also based on how specific DB system works: if hash table finds one entry and goes above and below in the corresponding data record file to find the rest of the records, it is also good, too. But if each and every record will be tracked from its data entry in the hash table, assuming there is no caching, there will be too many read I/O operation, undermining the effort.
Part.price	Unclustered	Both hash table and B+ tree would work: 1) based on implementation, hash table may work bad if bucket approach is not used 2) B+ tree would work better, as records having their city field as "Aachen" will be placed consecutively if data records are stored (instead of data entries)

(c) Suppose that the following additional information is available: The system's statistics indicate that there are suppliers from 100 cities, and part prices range from 1 to 1000 Euros. All attribute values are uniformly distributed and each id has been created in numerical order without any records deleted so far. There are a total of 400 suppliers, 2.000 parts, and 200.000 entries in the Supply relation in the database. For each of the query's base relations (Supply, Suppliers, Parts) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.

Here are the estimations:

- $(\sigma_{city="Aachen"} Suppliers)$
  #Entries without Selection = 400,
  #Entries with Selection = 400 * $\frac{1}{100}$ = 4

- $(\sigma_{pid<800} Supply)$
  #Entries without Selection = 200000,
  #Entries with Selection = 800,
  (note that pid is the primary key, and it is noted in the question that each ID is created in numerical order and none has been deleted)

- $(\sigma_{price<=200} Parts)$
  #Entries without Selection = 2000,
  #Entries with Selection = 2000 * $\frac{200-0}{1000-0}$ = 400

(d) Based on the results for (c), which pair of relations should be joined first?
As we want to avoid cross product as because of its cost, we want to look at which join routes are possible:

- (Suppliers $\bowtie$ Supply)
  Selection-applied Suppliers relation has 4 entries,
  Selection-applied Supply relation has 800

- (Supply $\bowtie$ Parts) Selection-applied Supply has 800 entries,
  Selection-applied Parts relation has 2000 entries.

Based on those numbers, it will be better to do (Suppliers $\bowtie$ Supply) join operation first; as it will reduce the number of 800 Supply entries to a hopefully far less qualifying matches of entries.
Instead of comparing 800 entries with 2000 entries in (Supply $\bowtie$ Parts), doing the first will make things run faster.

So, the answer is (Suppliers $\bowtie$ Supply) as it is expected to lead to a greater reduction in the number of operations.