1 SOLUTION::

1.1

Let us assume that R,S and T are sets and not bags. Hence, repetition of tuples in the relations is

not allowed.

Let R consist of attributes = R(a,b)
Let S consist of attributes = S(b,c)
Let T consist of attributes = T(b,c)

Given that it has to be proven that $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

LHS:

Let S and T consist of m and n tuples respectively. Then the total number of tuples after the union

operation (name the number of tuples as "st") is less than or equal to m+n, i.e, st <= m+n. In a union

operation, the tuples which are present in S and T, minus the common tuples are given as the output.

Now, S U T = S U T(b,c) where the number of tuples are st.

When a join operation is performed, there is a sharing attribute b using which every tuple r from R is matched with st and the joined tuple is given out. The number of tuples r of R may or may not be equal to st.

The output is as follows : RST (a,b,c)

RHS:

The output of the join operation between R and S is as follows: $R(a,b) \bowtie S(b,c) = RS(a,b,c)$

The output of the join operation between R and T is as follows: $R(a,b) \bowtie T(b,c) = RT(a,b,c)$

Now the union operation considers all the tuples of RS and RT minus the tuples which have repeated.

The output is as follows: RS U RT = RST (a,b,c)

Hence, it has been proven that the tuples obtained on the left hand side are equal to the tuples

obtained on the right hand side.

1.2

Let R contain the attributes = R(a,b)
Let S contain the attributes = S(b,c)

LHS:

The join operation of R and S gives out the table RS which contains the following attributes = RS(a,b,c). For every tuple in R, it is matched with a sharing attribute b with S and the joined tuples are obtained.

RHS:

(R semijoin S) gives out the tuples of R which have a matching tuple in S for a shared attribute "b".

The output given out can be classified as a shorter version of the relation R itself i.e., R(a,b).

Now (R ⋈ S) gives out combined tuple of R and S which have been joined using the sharing attribute "b". Hence, (R ⋈ S) = RS(a,b,c).

It can be noted that (R semijoin S) simply acts as an intermediate optimization technique so that all the tuples of R need not be compared to the tuples of S using the shared attribute when the main join operation is performed. The semi-join operation reduces the computation by comparing only those tuples of R that already have a shared value with tuples in S.

Thus, both the left hand side and right hand side give out similar kinds of tuples.

2 SOLUTION ::

Given query:

```
SELECT customer
FROM Likes cus
WHERE NOT EXISTS
(SELECT Often.customer
FROM Often, Provide
WHERE Cus.customer = Often.customer AND Often.club = Provide.club AND NOT EXISTS
(SELECT *
FROM Likes
WHERE Likes.customer = Often.customer AND Likes.activity = Provide.activity))
```

The query gives out the Customers who do not go to clubs often that do not provide the activities that they like.

Now, we can eliminate correlations and un-nest the query.

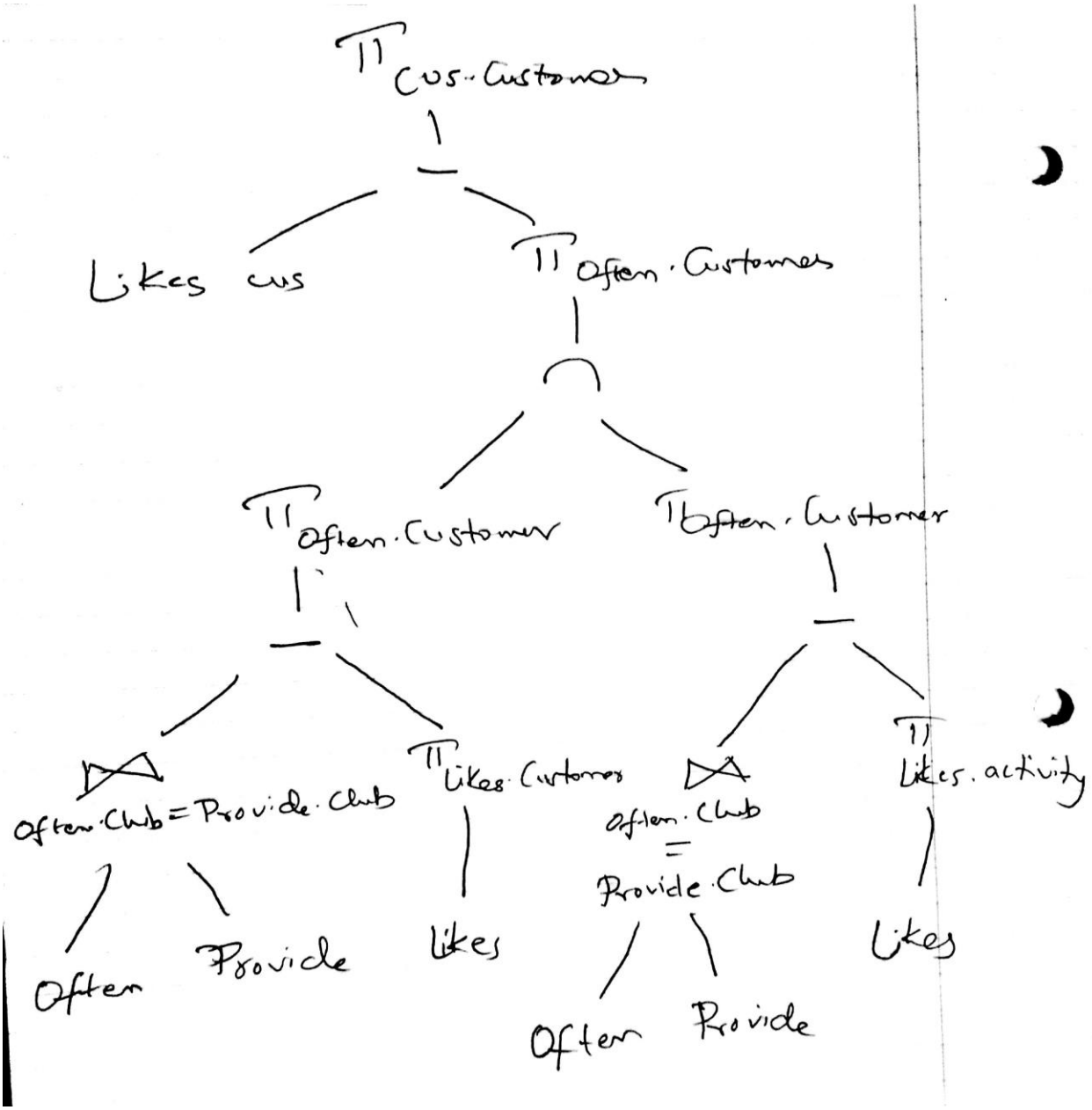
Intermediate query :

```
SELECT Cus.customer
FROM Likes cus
WHERE cus.customer NOT IN
(SELECT Often.customer
FROM Often, Provide
WHERE Often.club = Provide.club AND Often.customer, Provide.activity NOT IN
(SELECT Likes.customer, Likes.activity
FROM Likes) )
```

FINAL MODIFIED QUERY:

```
SELECT customer
FROM Likes cus EXCEPT (

((SELECT Often.customer
FROM Often, Provide
WHERE Often.club = Provide.club EXCEPT

(SELECT Likes.customer FROM Likes))

INTERSECT

((SELECT Often.customer

FROM Often, Provide

WHERE Often.club = Provide.club EXCEPT

(SELECT Likes.activity FROM LIKES))

)
```

The logical query plan is as follows:


Π cus.Customer
Likes cus
Π often.Customer
Π often.Customer
Π often.Customer
often.Club = Provide.Club
Π likes.Customer
often.Club = Provide.Club
Π likes.activity
often
Provide
likes
often
Provide
likes

3 SOLUTION::

Given that

T(Students) = 2000
T(University) = 100
T(Major) = 3000
B(Students) = 100
B(University) = 10
B(Major) = 100

Each school has a unique rank number (urank value) between 1 and 100 and there are 20 different cities.

We compute the operations in the following way:

For (1) : B(Students) + B(Students) * 1/ V(Students, city) = 100 + 100 * 1/20 = 105

For (2) : Using the selectivity factor for range queries:

Here c =10 and low(University, urank) = 1 and high(University, urank) = 100 and T(University) = 100.

The formula is

*C – low(University, urank) / high(University, urank) - low(University, urank) * T(University)*

Therefore, I/O for (2) = 9 tuples.

For (3) : 2 I/Os

For (4): Since the number of tuples are 9, the I/O for index nested loop = 9.

For (5) and (6) : 0, since they are performed on the fly.

Thus, the total I/Os are : 105+2+9+9+0+0 = 125.

4 SOLUTION ::

It is given that R and U each have 1000 tuples, while S and T each have 100 tuples. Also,

V (R; a) = V (R; b) = V (S; b) = V (T; d) = V (U; d) = V (U; a) = 100, and V (S; c) = V (T; c) = 10.

R(a,b)	S(b,c)	T(c,d)	U(d,a)
V(R,a) = 100			V(U,a) = 100
V(R,b) = 100	V(S,b) = 100		
	V(S,c) = 10	V(T,c) = 10	
		V(T,d) = 100	V(U,d) = 100

Since, the single relation does not have any intermediate relations needed and the best expression is the relation itself, the cost for this singleton set is 0. The table below shows the individual relation costs:

	R	S	T	U
Size	1000	100	100	1000
Cost	0	0	0	0
Best Plan	R	S	T	U

Now, let's consider the pairs of relations. The cost is still 0 since there are no intermediate relations in a join of two relations.

The formula to calculate size is : T(R) * T(S) / max(V(R,b), V(S,b)) . This standard formula is used to compute all the sizes.

The table for the join is as shown below:

	{ R , S }	{ R , T }	{ R , U }	{ S , T }	{ S, U}	{ T, U}
Size	1000	1,00,000	10,000	1,000	1,00,000	1000
Cost	0	0	0	0	0	0
Best Plan	R ⋈ S	R ⋈ S	R ⋈ T	S ⋈ T	S ⋈ U	T ⋈ U

Now, consider the join of three relations. Let us consider {R,S,T}. The pairs of joins among R,S and T are considered and the least one is picked i.e., R ⋈ S = 1000 or S ⋈ T =1000, but we pick R ⋈ S. Also here the size of the intermediate relation from the previous join is considered as the cost for this join.

	{ R,S,T }	{ R, S, U }	{ R, T, U }	{ S, T , U }
Size	10,000	10,000	10,000	10,000
Cost	1000	1,000	1,000	1,000
Best plan	(R ⋈ S) ⋈ T	(R ⋈ S) ⋈ U	(T ⋈ U) ⋈ R	(S ⋈ T) ⋈ U

Grouping	Cost
(R ⋈ S) ⋈ T) ⋈ U	11,000
(T ⋈ U) ⋈ R) ⋈ S	11,000
(R ⋈ S) ⋈ U) ⋈ T	11,000
(S ⋈ T) ⋈ U) ⋈ R	11,000

If we consider left deep trees then the cost involved is 11,000 for all of them. But if we include bushy trees in our dynamic programming strategy then the join of R and S joined with the join of T and U yields the lowest cost of 2,000.

Grouping	Cost
(R ⋈ U) (S ⋈ T)	11,000
(R ⋈ S) (T ⋈ U)	2,000
(R ⋈ T) (S ⋈ U)	2,00,000

Hence, this option can be picked.

REFERENCES and RESOURCES:

1. Database Systems, The complete book.
2. https://msdn.microsoft.com/en-us/library/ms188055.aspx
3. Lecture slides.

DISCUSSION:

This assignment was discussed with the following people:

1. Rohit Nedunuri
2. Prerit Anwekar