1: Query Optimization

Consider the following relational schema and SQL query:
Suppliers(sid,sname,address,category)
Supply(sid,pid)
Parts(pid,pname,brand)

```
SELECT S.sname, P.pname
FROM Suppliers S, Parts P, Supply Y
WHERE sS.sid = Y.sid AND Y.pid = P.pid
```

1. Give two examples of join orderings that System R will not consider.

   Solution:

   System R optimizer does not consider join orderings that require the computation of cross-products to calculate this join. Further, it considers only left-deep joins. The followings are two join orderings that are not left-deep and/or contain Cartesian product.

   - P $\bowtie$ (S $\bowtie$ Y) (right-deep join)
   - (S $\bowtie$ P) $\bowtie$ Y (Cartesian product)

2. Enumerate all joins orderings that System R optimizer considers.

   Solution:

   System R optimizer does not consider join orderings that require the computation of cross-products to calculate this join. Further, it considers only left-deep joins. Thus, ot considers the following join orderings:

   - (S $\bowtie$ Y) $\bowtie$ P
   - (Y $\bowtie$ S) $\bowtie$ P
   - (P $\bowtie$ Y) $\bowtie$ S
   - (Y $\bowtie$ P) $\bowtie$ S

2: Query Optimization

Consider relations R(A,B) and S(B,C). Assume that R contains 2,000 tuples, and that s contains 5,000 tuples. We want to compute the equi-join $R \bowtie S$ over attribute B.

1. Without any further assumptions, what is the maximum number of tuples that the resulting relation may contain?

   Solution:

   The join will have the maximum number of tuples if all values of attribute B in R and S have equal values, therefore, the maximum number of tuples is 2000 $\times$ 5000 = 10,000,000.

2. Now assume that we know that the number of distinct values of B in R is 500. What is now a reasonable estimate on the size of join?

   Solution: 2000 $\times$ 5000 / 500 = 20,000.

3. Finally, assume we know that B is a primary key in S. What is now a reasonable estimate on the size of join?

   Solution:

   If B is the primary key of S, it will have 5,000 distinct values in S. Hence, the size of join will be:

   2000 $\times$ 5000 / max(500,5000) = 2,000.

3: Query optimization

Consider the following relations:

Movie (Title, Year, Rating, StudioName)
Studio (Name, Country, Address)

Assume each movie is produced by just one studio, whose name is mentioned in the *StudioName* attribute of the Movie relation. Attributes *Title* and *Name* are primary keys for relations *Movie* and *Studio*, respectively. Attribute *StudioName* is a foreign key from relation *Movie* to relation *Studio*. Attribute *Rating* shows how popular a movie is and its values are between 1-10. The following statistics are available about the relations.

Movie	Studio
T(Movie) = 24000 V(Movie,StudioName)=800 V(Movie,Rating)=10	T(Studio)= 1000

The following query returns the movies with a rating of 10 produced in each country after 1990.

```
SELECT Country, Title
FROM Movie, Studio
WHERE Movie.StudioName = Studio.Name and Year > 1990 and Rating = 10
```

Suggest an optimized logical query plan for the above query. Then, estimate the size of each intermediate relation in your query plan. By an intermediate relation, we mean the relation created after each selection or join.

Solution:
The optimized logical query plan for the above query is the following. The plan first selects table *Movie* according to the conditions on *Year* and *Rating*. It then projects *Movie* on *StudioName* and *Title*. It also projects table *Studio* on attributes *Name*, *Country*. Finally, it joins the projected *Movie* and *Studio* relations and projects the resulting table on attributes *Country* and *Title*.

There are two relations created after the selection and the join. The size of the relation created after the selections over *Movie* is 24000/(10 $\times$ 3)=800 . The size of the relation created after the join is (800 $\times$ 1000)/1000=800.

4: Serializability and 2PL

Consider the following schedule.

	T1	T2	T3
0	start		
1	read B		
2	write B		
3		start	
4		read B	
5		write B	
6			start
7			read A
8			write A
9		read A	
10		write A	
11		COMMIT	
12	read D		
13	write D		
14			COMMIT
15	COMMIT		

(a) Give the serial schedule that this schedule is equivalent to. If none, explain why.

Solution: The serializability graph for the above schedule is: T1 $\rightarrow$ T2 $\leftarrow$ T3. Any order that complies with the topological order of the graph, such as T1 $\rightarrow$ T3 $\rightarrow$ T2 or T3 $\rightarrow$ T1 $\rightarrow$ T2, is an equivalent serial schedule for our schedule.

(b) Explain whether this schedule is consistent with two phase locking (2PL)? Assume that in 2PL, each transaction requests a proper lock, i.e., S lock for read and X lock for write, just before executing the operation. If your answer is yes, insert into the schedule a minimal set of additional operations that will make the schedule no longer consistent with two phase locking. Do not introduce any new transaction. If your answer is no, then remove from the schedule a minimal set of operations, so that the revised schedule is consistent with two phase locking.

Solution: This schedule is **not** 2PL because when T1 has the X lock on B, T2 has to get S lock on B to read it. T1 cannot release the lock on B because it has to request and get S lock on D before time step 12. Thus, releasing X lock on B violates 2PL rules. One can remove the read and write operations of T1 on B to create a schedule that complies with 2PL.

(c) Consider the version of the schedule that is consistent with two phase locking (either the original one or your revised version, depending on your answer to the previous part). Is that version consistent with strict two phase locking, where each transaction release all locks just after commit? Why or why not?

Solution: The revised schedule produced in the previous part is **not** strict 2PL because T3 has to release its lock on A before commit.

2: Locking

Consider the following locking protocol. Before a transaction T writes a data object A, T acquires an exclusive lock on A, and holds onto this lock till the end of transaction. Before a transaction T reads a data object A, T acquires a shared lock on A, but releases the lock immediately after reading A. Write a schedule where each transaction follows this locking protocol, but the schedule is not serializable.

Solution: Consider the following schedule. Each transaction in the schedule follows the proposed locking protocol. Its serialization graph contains edges T1 $\rightarrow$ T2 and T2 $\rightarrow$ T1, therefore it is not serializable.

	T1	T2
0	start	
1	S.lock (A)	
2	read A	
3	S.releaseLock(A)	
4		start
5		X.lock(A)
7		write A
6		COMMIT
7		X.releaseLock(A)
8	S.lock(A)	
9	read A	
10	S.releaseLock(A)	
11	COMMIT	

3: Multi-granularity locking

Consider a database organized in terms of the following hierarchy of objects: The database itself is an object (D), and it contains two files (F_1 and F_2), each of which contains 1000 pages ($P_1, \cdots,$ P_{1000} and $P_{1001}, \cdots, P_{2000}$, respectively). Each page contains 100 records, and records are identified as p : i, where p is the page identifier and i is the slot of the record on that page. Multiple-granularity locking is used, with S, X, IS, IX and SIX locks, and database level, file-level, page-level and record-level locking. For each of the following operations, indicate the sequence of lock requests that must be generated by a transaction that wants to carry out (just) these operations:

1. Read record P_{1200} : 5.
2. Read records P_{1200} : 98 through P_{1205} : 2.
3. Read all (records on all) pages in file F_1.
4. Read pages P_{500} through P_{520}.
5. Read pages P_{10} through P_{980}.
6. Read all pages in F_1 and (based on the values read) modify 10 pages in F_1.

7. Delete record P_{1200}:98. (This is a write.)
8. Delete the first record from each page. (Again, these are writes.)

solution:
The answer to each question is given below.

1. IS on D; IS on F_2; IS on P_{1200}; S on P_{1200}:5.
2. IS on D; IS on F_2; IS on P_{1200}, S on 1201 through 1204, IS on $P_1$205; S on P_{1200}:98/99/100, S on P_{1205}:1/2.
3. IS on D; S on F_1.
4. IS on D; IS on F_1; S on P_{500} through P_{520}.
5. IS on D; S on F_1 (performance hit of locking 970 pages is likely to be higher than other blocked transactions).
6. IS and IX on D; SIX on F_1.
7. IX on D; IX on F_2; X on P_{1200}. (Locking the whole page is not necessary, but it would require some reorganization or compaction after the delete.)
8. IX on D; X on F_1 and F_2. (There are many ways to do this, there is a trade-off between overhead and concurrency.)