

ASSIGNMENT 4 - 600.315/415 - Databases

Date of Last Acceptance Tuesday, December 6, 3 PM

- (1) Consider the following relational schema and SQL query (next page).

EMPLOYEE	<u>SSN</u>	Name	DNO	Salary	Hobby	Sex
	339-18-4886	Milton Kent	520	65000	Golf	M

DEPARTMENT	<u>DNO</u>	DName	Budget	Location	MGRSSN
	520	Accounting	10000000	NEB	339-18-4886

WORKS_ON	<u>ESSN</u>	<u>PNUM</u>
	339-18-4886	109

PROJECT	<u>PNUM</u>	PName	Budget	Location	Goal
	520	F16Engine	30000000	NEB	Profit

Assume that there are 50,000 employees, 1,000 projects, 30 departments 50% of employees are male, 0.01% of employees have a yodeling hobby, the Relational Mechanics department only has 6 members and the median project budget is \$1,000,000. 5 people work on the F16Engine project.

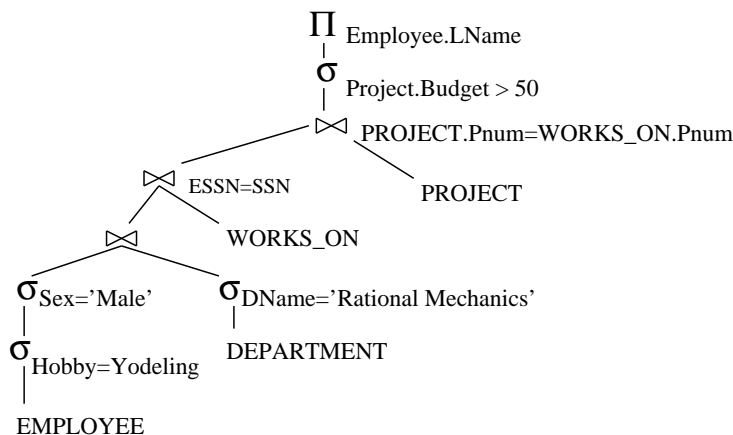
- (a) Convert the following SQL query into the relational algebra and draw a query tree for it for the most efficient execution ordering you can think of.

```

SELECT E.LName
FROM   Employee E, Department D, WorksON W, Project P
WHERE  E.DNO=D.DNO and E.SSN=W.ESSN and P.PNUM=W.PNUM
       and P.Budget > $50 and E.Sex='M' and E.hobby='Yodeling'
       and D.Dname='Rational Mechanics';
    
```

Answer: Relational algebra:

$$\prod_{Lname} (\sigma_{budget > 50 \text{ and } sex = 'male' \text{ and } hobby = 'yodeling' \text{ and } dname = 'Rational Mechanics'}$$

$$(\text{Employee} \bowtie \text{Department} \bowtie \text{Works_on} \bowtie \text{Project}))$$


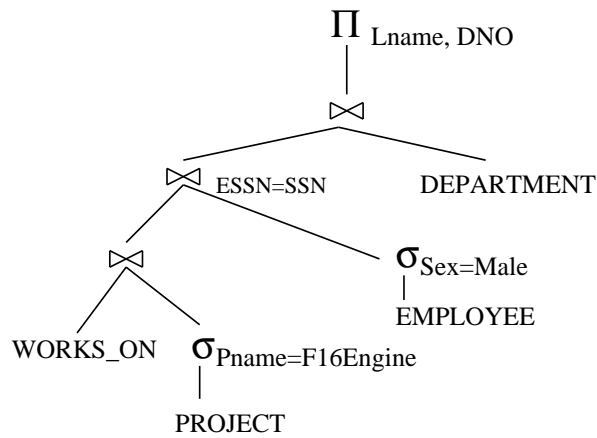
- (b) Convert the following SQL query into the relational algebra and draw a query tree for it for the most efficient execution ordering you can think of.

```

SELECT E.LName, D.DNO
FROM   Employee E, Department D, WorksON W, Project P
WHERE  E.DNO=D.DNO and E.SSN=W.ESSN and P.PNUM=W.PNUM
      and P.Fname='F16Engine' and E.Sex='M'

```

Answer: Relational Algebra:

$$\Pi_{lname, dno} (\sigma_{fname='f16engine' \wedge sex='M'} (Employee \bowtie Department \bowtie Work_on \bowtie Project))$$


(2) Consider the following two transactions:

T_0	T_1
read_item(A)	read_item(B)
read_item(B)	read_item(A)
if A=0 then B :=B+1	if B=0 then A := A+1
write_item(B)	write_item(A)

(a) Show a concurrent execution of T_0 and T_1 which produces a serializable schedule.

Answer: There is no concurrent execution schedule that is conflict serializable where any instruction of T_1 overlaps with T_0 . To demonstrate this, consider putting only the first instruction of T_1 before only the last instruction of T_0 , or putting only the first instruction of T_0 before only the last instruction of T_1 and it should be clear to you that there is a cycle.

However, all serial schedules are trivially concurrent (one does not need to have overlapping instructions to be concurrent). Thus the following schedule is serializable because it is already serial. There is no precedence cycle (give a precedence graph to convince yourself of this).

T_0	T_1
read_item(A)	read_item(B)
read_item(B)	read_item(A)
if A=0 then B :=B+1	if B=0 then A := A+1
write_item(B)	write_item(A)

(b) Show a concurrent execution of T_0 and T_1 which produces a non-serializable schedule

As noted above, any concurrent schedule with an overlap is not conflict serializable. Below is one. Draw a precedence graph to convince yourself of this.

T_0	T_1
read_item(A)	read_item(B)
read_item(B)	read_item(A)
if A=0 then B :=B+1	if B=0 then A := A+1
write_item(B)	write_item(A)

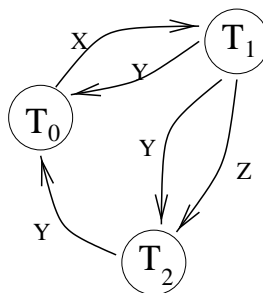
- (3) Consider the following 3 transactions, consisting of a sequence of read and write operations, with (omitted) intermediate code.

T_0	T_1	T_2
read_item(X)	read_item(Z)	read_item(Y)
write_item(X)	read_item(Y)	read_item(Z)
read_item(Y)	write_item(Y)	write_item(Y)
write_item(Y)	read_item(X)	write_item(Z)
	write_item(X)	

Now consider the following concurrent schedule of these 3 transactions. Is this schedule conflict serializable? Why or why not (create a labeled precedence graph to help you argue). If it is conflict serializable, give an equivalent serial schedule.

T_0	T_1	T_2
	read_item(Z)	
	read_item(Y)	
	write_item(Y)	
read_item(X)		read_item(Y)
write_item(X)		read_item(Z)
		write_item(Y)
		write_item(Z)
read_item(Y)	read_item(X)	
write_item(Y)		
	write_item(X)	

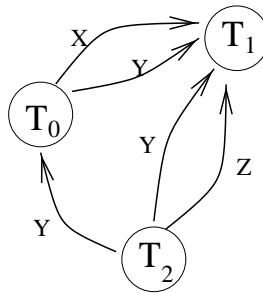
This concurrent schedule is not conflict serializable because a cycle exists in its precedence graph (shown below):



- (4) Consider the 3 transactions given in problem 3, along with the following concurrent schedule. Is this schedule conflict serializable? Why or why not (create a labeled precedence graph to help you argue). If it is conflict serializable, give an equivalent serial schedule.

T_0	T_1	T_2
read_item(X) write_item(X)		read_item(Y) read_item(Z)
	read_item(Z)	write_item(Y) write_item(Z)
read_item(Y) write_item(Y)	read_item(Y) write_item(Y) read_item(X) write_item(X)	

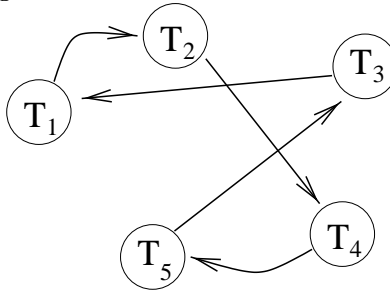
This concurrent schedule is conflict serializable because no cycle exists in its precedence graph (shown below):



One equivalent serial schedule would be: $T_2 \rightarrow T_0 \rightarrow T_1$, which is obtainable from a topological sort. Below this is shown in more detail (*not* required for an exam) - convince yourself that this gives identical output to the original overlapping concurrent schedule, i.e. that swapping the order of instructions in this way did not affect the outcome.

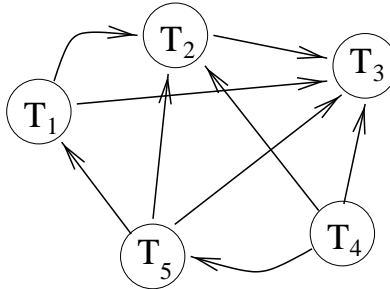
T_0	T_1	T_2
		read_item(Y) read_item(Z) write_item(Y) write_item(Z)
read_item(X) write_item(X) read_item(Y) write_item(Y)	read_item(Z) read_item(Y) write_item(Y) read_item(X) write_item(X)	

- (5) Consider the precedence graph below. Is the corresponding transaction history conflict serializable? If so, give an equivalent valid serial schedule for the transactions.



No, the transaction is not conflict serializable because there is a cycle in the precedence graph

- (6) Consider the labeled precedence graph below. Is the corresponding transaction history conflict serializable? If so, give an equivalent valid serial schedule for the transactions.



Yes, the schedule is conflict serializable. There is no cycle, and the only valid topological sort is:

$T_4 \rightarrow T_5 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$

- (7) Consider a relation that is fragmented horizontally by *PlantLocation*:

EMPLOYEE	Name	Title	Salary	PlantLocation
	Joe Steel	Foreman	65000	Edmonton

Assume each fragment has two replicas: one stored at the New York office and one stored locally at each plant location. Describe a good processing strategy for the following queries entered at the San Jose plant location.

- (a) Find all employees at the 'Boca' plant.
- (b) Find the average salary of all employees in the company.
- (c) Find the highest-paid employee at each of the following locations Toronto, Edmonton, Vancouver, Montreal.
- (c) Find the lowest paid employee in the entire company.

Answer: a:

- (1) Send the query $\Pi_{name} Employee$ to the Boca plant
- (2) After processing at Boca plant, have the boca location send back the answers to san Jose

b:

- (1) Send the query to New York
- (2) After computing average at New York, send answer back to San Jose

c:

- (1) Send the query to Toronto, Edmonton, vancouver, Montreal
- (2) Compute the queries at those sites locally
- (3) Send answer back to San Jose

d:

- (1) Send the query to New York
- (2) Compute the query (find the lowest salaries employee) at New York
- (3) Send answer back to San Jose

(8) Consider the relations:

EMPLOYEE	Name	Title	Salary	PlantLocation
	Joe Steel	Foreman	65000	Edmonton

MACHINE	MachineNumber	Type	PlantLocation
	117	Infusinator	Edmonton

Assume the EMPLOYEE relation is fragmented horizontally by PlantLocation and each fragment is store locally at its corresponding plant site. Assume the MACHINE relation is stored in its entirety at the Armonk site. Describe a good strategy for processing each of the following queries.

- Find all employees at the plant containing machine number 1130.
- Find all employees at plants containing machines whose type is “Milling Machine”
- Find all machines at the Almaden plant.
- Compute $EMPLOYEE \bowtie MACHINE$.

Answer: a:

- Compute $\prod_{plantlocation} (\sigma_{MachineNumber=1130} MACHINE)$ at Armonk site
- Send query $\prod_{name} (Employee)$ to all sites returned from previous query
- Compute at those sites locally
- Union the answers at the destination site

b: The same as above (8a), except the first query should change to

$$\prod_{planLocation} (\sigma_{type='MillingMachine'} MACHINE)$$

c:

- Compute $\prod_{type} (\sigma_{plantLocation='Armonk'} (MACHINE))$ at Armonk
- Send the answer to the destination site

d:

- Group MACHINE into subrelations with the same plantlocation at Armonk
- Send the sub-relations to the corresponding plantlocation
- Compute a join between Employee(local data) and Machine subrelaiton (sent from Armonk)
- Union the results at the destination site

- (9) Compute $r \bowtie s$ and $s \bowtie r$ for the following relations:

r:	A	B	C
	9	1	8
	2	3	5
	6	4	3
	5	6	7
2	3	4	

s:	C	D	E
	4	5	6
	3	4	3
	1	3	4
	1	5	2
4	7	9	

$r \bowtie s:$	A	B	C	D	E
	2	2	4	5	6
	2	3	4	7	9
	6	4	3	4	3

$s \bowtie r:$	C	D	E
	4	5	6
	3	4	3
	4	7	9

- (10) If a hash structure is used on a search key for which range queries are likely, what property should the hash function have?

Answer: The hash function should preserve the order of the data. That is, if h is the hash function, then $x \leq y$ must imply $h(x) \leq h(y)$.

- (11) Recall that the hash join algorithm (for the natural *inner* join $A \bowtie B$) basically adds the join attribute of B to the hash table, then hashes on A into the same table. If a match is found, then the join on that attribute succeeds. If no match is found, the join on that attribute does not succeed, but the attribute of A is *not* added to the hash table.

Briefly list the changes necessary to this algorithm for to handle a full *outer* join.

Answer: 1. If t_a and t_b match, $t_a \bowtie t_b$ is added to the result and t_b tuples in the input are marked as covered. 2) if no match is found for t_a , then pad it with NULLs and add to the result. 3). After matches have been attempted for all the tuples in t_a , then all the unmarked tuples in t_b are padded with NULLs and added to the result

- (12) Suppose that the bank used in the textbook's running example maintains a statistical database containing the average balances of all customers. The scheme for this relation is $(customer\text{-}name, customer\text{-}city, avg\text{-}balance)$. Assume that for security reasons, the following restrictions are imposed on queries against this data:

- Every query must involve at least 10 customers
- The intersection of any pair of queries must be at most 5.

Construct a series of queries to find the average balance of a customer. (Hint: this can be done in fewer than 7 queries.)

Answer: Let X_i be the *avg-balance* of customer i . Execute 4 queries Q_1, Q_2, Q_3, Q_4 .

$$Q_1 : \sum x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} \quad Q_2 : \sum x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}$$

$$Q_3 : \sum x_1, x_2, x_3, x_4, x_5, x_6, x_{12}, x_{13}, x_{14}, x_{15}, x_{16} \quad Q_4 : \sum x_1, x_7, x_8, x_9, x_{10}, x_{11}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}$$

Then,

$$X_1 = \frac{(S_3 + S_4) - (S_1 + S_2)}{2}$$