Section 1: Redundancy Anomalies [13 points]

1- (9 points) Consider the following table. Give an example of update anomaly, an example of deletion anomaly and an example of insertion anomaly knowing that different salesmen can sell the same car but each salesman has a different commission. The commission for one salesman is always the same. The discount depends upon the date.

Car #	Date-sold	Salesman	Commission	Discount
123	2003-02-13	35	3%	10%
321	2003-01-31	45	5%	15%
123	2003-02-13	20	2.5%	10%
918	2003-01-18	45	5%	5%
789	2003-02-10	19	3%	5%

Update Anomaly:

- Changing the commission of a salesman requires changing all tuples related to the salesman

- Changing the discount for a day requires updating all records of that particular day

Deletion Anomaly:

- By deleting the only sale by a salesman we loose the information about the commission for that salesman

- By deleting the only sale of the day we loose the information about the discount given that day

Insertion Anomaly:

- We can’t add a salesman until a sale is done

- We can’t add a discount information until a sale is done that day

2- (4 points) Give a schema of a decomposition that avoids such anomalies.

Sales(Car#, Date-sold, Salesman)
Salesmen(Salesman, Commission)
Discounts(Date, Discount)

Section 2: Concurrency Control [21 points]

[5 points] Briefly explain what is Atomicity and enumerate the other remaining ACID properties. You don't have to explain C and I, but explain the term associated with D.

```
Atomicity: All or nothing: All operations of a transactions are
executed or none.   (2 points)

C stands for   Consistency   (0.75 point)
I stands for   Isolation     (0.75 point)

D stands for   Durability   : The effect of a committed
transaction should persist even after a crash. (1.5 point)
```

1- [4 points] Given the following schedule S:

T1:		R(Y);		W(X);
T2:	R(Y);		R(X); W(Y);	

Is S a serial schedule? Explain why.
Give a serial schedule equivalent to S.

```
No, it is a non-serial schedule since transactions are
interleaved. (2 points)

There is no serial schedule equivalent to S. S is not
serializable. There is a cycle in the dependency graph.
(2 points)
```

2- [12 points] Assume the following actions listed in the order they are scheduled and prefixed with the transaction name. Assume that the timestamp of a transaction *Ti* is *i*. T1:R(Y), T2:R(X), T3:R(Y), T1:R(X), T1:W(Y), T2:W(X), T3:R(X)
Add lock and unlock requests and describe how the following concurrency control mechanism A, B and C handle the sequence by giving the schedule with waiting time between actions. The DBMS should process the actions in the order shown. If a transaction is blocked it waits and its actions are queued until it resumes. When a transaction waits, the DBMS continues with the next action of an unblocked transaction in the sequence.

A- Strict 2PL with deadlock detection
B- Strict 2PL with timestamps used for deadlock prevention with Wait-Die policy
C- Strict 2PL with timestamps used for deadlock prevention with Wound-Wait policy.

T1:R(Y), T2:R(X), T3:R(Y), T1:R(X), T1:W(Y), T2:W(X), T3:R(X)

A (Strict 2PL)			**B** Strict 2PL + Wait-Die			**C** Strict 2Pl+Wound-Wait		
T1	T2	T3	T1	T2	T3	T1	T2	T3
X(Y)			X(Y)			X(Y)		
R(Y)			R(Y)			R(Y)		
	X(X)			X(X)			X(X)	
	R(X)			R(X)			R(X)	
		S(Y)			S(Y)			S(Y)
		Wait			Abort			wait
S(X)			S(X)			S(X)		
Wait			wait				Abort	
	W(X)				S(Y)	R(X)		
	UL(X)				Abort		X(X)	
R(X)				W(X)			wait	
W(Y)				UL(X)		W(Y)		
UL(Y)					S(Y)	UL(Y)		
UL(X)					Abort	UL(X)		
		R(Y)	R(X)					R(Y)
		S(X)			S(Y)		R(X)	
		R(X)			Abort			S(X)
		UL(Y)	W(Y)					wait
		UL(X)	UL(Y)				W(X)	
			UL(X)				UL(X)	
					R(Y)			R(X)
					S(X)			UL(Y)
					R(X)			UL(X)
					UL(Y)			
					UL(X)			

A: **(4 points)**

B: **(4 points)**

C: **(4 points)**

R(X) means read X
W(X) means write X
X(X) means request an exclusive lock on X
S(X) means request a shared lock on X
UL(X) means release the lock (unlock) X

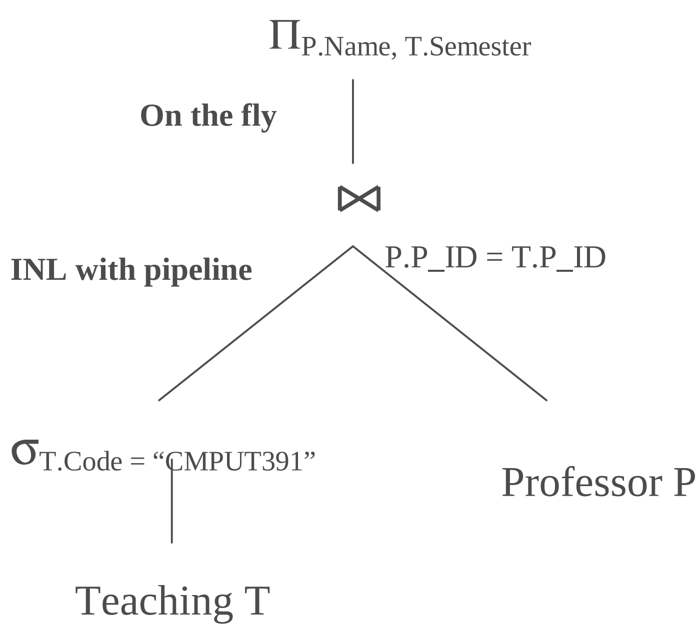
Section 3: Query Optimization [40 points]

Given the following relations for the entities Professor and Course and the relationship Teaching:

Professor (P_ID, Name, Dept_ID)
Course(Code, Dept_ID, CName, syllabus)
Teaching(P_ID, Code, Semester)

1- [8 points] Given the following SQL query Q, draw the query plan with an early selection and an index nested loops join strategy knowing that the relation Professor is indexed on P_ID. How relevant is it that the index on Professor is clustered or not?

SELECT P.Name, T.Semester
FROM Professor P, Teaching T
WHERE P.P_ID = T.P_ID **AND**
T.Code = "CMPUT391"



(4 points for the query tree)
(2 points for the plan)

Clustered or not is irrelevant since for each tuple in the outer relation we would need to access a tuple in the inner relation using the index. **(2 points)**

2- [32 points] Would it be better to do a bloc nested loops join or a join that takes into account the index for Professor? Suppose we have 4 buffer pages (blocs) in main memory, and assuming a uniform distribution for all the values in the database, estimate the query execution I/O cost for these two plans. There are 1000 courses. The relation Course is contained in 200 pages of disk, each page with 5 tuples of Course. There are 50000 teaching records. The relation Teaching is contained in 5000 pages, each page with 10 tuples of Teaching. There are 105 professors. The relation Professor is contained in 14 pages, each page with up to 8 tuples. There is no index on attribute Code for the relation Teaching and the relation is neither clustered nor sorted on this attribute. Professor is indexed on P_ID using a hash index.

INL **(16 points)**
Selection needs 5000 I/O
The size of the result is 50000/1000 = 50 teaching for CMPUT391 = 5 pages

Cost of join is 50 * 1.2 = 60 I/O
Total = 5060 I/O

BNL **(16 points)**
Selection needs 5000 I/O
Join = ⌈5/3⌉ * 14 = 28 I/O
Total 5028 I/O

Section 4: Functional Dependencies [26 points]

1- [4 points] Consider the following set F of functional dependencies, find the projection of F onto AFE. A→ BC, C→FG, E→HG, G→A.

```
F+ :                                    F_AFE
  A→A                                     A→A  }
  B→B                                     F→F  } (1 point)
  C→C                                     E→E  }
  F→F                                     A→F  }
  G→G                                     E→A  } (2.5 points)
  E→E                                     E→F  }
  H→H                                     E→AF   (0.5 point)
  A→B
  A→C
  C→F => A→F
  C→G
  A→FG
  E→HG
  E→H, E→G
  G→A => E→A
…
```

2- [6 points] Consider a relation with attributes A, B, C, D, and E and the functional dependencies B→EC, C→A, A→D and D→E. Show that the decomposition of the schema into AB, BCD and ADE is lossless. Is this decomposition dependency preserving?

A→D and D→E => A→DE => A is key in ADE
(ADE) ∩ (AB) = A which is key in ADE
=> ADE and AB is loss-less join decomposition
(2 points)

B→EC, since C->A and A→D then B→A and B→D
=> B→ABCDE B is key

(BCD) ∩ (AB) = B which is key
➔ BCD and AB is lossless join decomposition
(2 points)

This is not dependency preserving since B→EC and C→ A are not preserved. **(2 points)**

Can also be verified with $(F_{AB} \cup F_{ADE} \cup F_{BCD})^+$

3- [10 points] Consider the relation R(A, B, C, D, E) with the following dependencies:
AB→C, CD→E, DE→B
Could AB be a key for this relation? Explain, and if AB is not a key, is ABD a candidate key? Explain. Could we justify the use of ADE as a key? Show it using inference rules.

AB+ = {A,B,C} => not a key since we don't have all attributes. (3 points)

ABD+ = {A,B,C,D,E} => ABD is a candidate key ABD→ABCDE (3 points)

1- DE→B
2- ADE→AB (augmentation)
3- AB→C
4- ADE→C (transitivity from 2 and 3)
5- ADE→ADE (trivial)
6- ADE→ ABCDE (from 2, 4, and 5)
yes ADE could be used as a key. (4 points)

4- [6 points] Decompose in 3NF the relation in question 3 above using the same functional dependencies and ADE as a key.

DE→B
violates 2NF since part of a key (DE) determines a non key (B)

(BDE) (ACDE) is in 3NF but not in BCNF