Section 1: Armstrong Axioms [10 points]

1- (6 points) State the three Armstrong Axioms (reflexivity, augmentation and transitivity) and demonstrate (argue) that they are sound:

We know: $X \rightarrow Y \Rightarrow t_1 \in r \ \& \ t_2 \in r$, if $\Pi_x(t_1) = \Pi_x(t_2)$, then $\Pi_y(t_1) = \Pi_y(t_2)$ (definition of functional dependency)

(1) Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$

Suppose R(A,B,C,D,E), X is {A,B,C,D} and Y is {C,D}. We have $Y \subseteq X$
Given $t_1=(a_1, b_1, c_1, d_1, e_1)$, $t_2=(a_2, b_2, c_2, d_2, e_2)$,
$\Pi_x(t_1) = \Pi_x(t_2) \Rightarrow c_1 = c_2 \ \& \ d_1 = d_2 \Rightarrow \Pi_y(t_1) = \Pi_y(t_2)$ (by definition)

(2) Augmentation : If $X \rightarrow Y$, then $XZ \rightarrow YZ$

Suppose R(A,B,C,D,E), X is {A,B} Y is {C,D} and Z is {E}.
Given $\Pi_{xz}(t_1) = \Pi_{xz}(t_2) \Rightarrow a_1 = a_2 \ \& \ b_1 = b_2 \ \& \ e_1 = e_2$
Since $X \rightarrow Y \Rightarrow$ if $(a_1 = a_2 \ \& \ b_1 = b_2)$ then $c_1 = c_2 \ \& \ d_1 = d_2$
So we have $c_1 = c_2 \ \& \ d_1 = d_2 \ \& \ e_1 = e_2$, thus $\Pi_{yz}(t_1) = \Pi_{yz}(t_2)$

(3) Transitivity: If $X \rightarrow Y \ \& \ Y \rightarrow Z$, then $X \rightarrow Z$

Suppose R(A,B,C,D,E), X is {A,B} Y is {C,D} and Z is {E}.
Gven $\Pi_x(t_1) = \Pi_x(t_2) \Rightarrow a_1 = a_2 \ \& \ b_1 = b_2$
$X \rightarrow Y \Rightarrow c_1 = c_2 \ \& \ d_1 = d_2$
$Y \rightarrow Z \Rightarrow e_1 = e_2$
Hence $\Pi_z(t_1) = \Pi_z(t_2)$

2- (4 points) Given the relation schema *R*(A, B, C, D, E) and the following set *F* of functional dependencies:

A → BC
CD→ E
B → D
E → A

List the candidate keys for *R*.

Hint: Compute the closure of F and find attributes that determine the whole set of attributes in R. You do not have to enumerate all F+ to find all candidate keys.

A→BC => A→B & A→C

A→B ^ B→D => A→D

A→CD ^ CD→E => A→E

A→A then A→ABCDE ←← A is a candidate key

E→A then E→ABCDE ←← E is a candidate key

CD→E then CD→ABCDE ←← CD is a candidate key

B→D then BC→CD then BC→ABCDE ←← BC is a candidate key

So candidate keys for R are A, E, CD and BC

Section 2: Database design [15 points]

1- [3 points] Given a relation schema *R*(A, B, C, D, E) and the following functional dependencies:
A→BC
CD→E
B→D
E→A
Show that the decomposition of *R* into (A, B, C) and (A, D, E) is a lossless-join decomposition. Notice that R and the functional dependencies are the same as in question 2 of the previous section.

Lossless Join Decomposition => R1∩R2 →R1 or R1∩R2→R2
In section1 question 2: A is a candidate key, so A→ ABCDE. Since R1∩R2=A and A→ABC , A→ ADE then the intersection determines not only one relation but both. So it is a lossless join decomposition.

2- [3 points] Show that the decomposition (A, B, C) and (C, D, E) of the relation R of the previous question with the functional dependencies also listed in the previous question is not a lossless-join decomposition.

It is not a lossless join decomposiotion because the intersection C is not a determinant for any of the relation schemas. Here is a counter example. It suffices that the value of C is repeated. For example:

R:

a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

R1:

a_1	b_1	c_1
a_2	b_2	c_1

R2:

c_1	d_1	e_1
c_1	d_2	e_2

R1∞R2:

a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

It is obvious that R1∞R2≠R

NB: ∞ is for the symbol of join.

3- [3 points] Is the decomposition of R into (A, B, C, E) and (B, D), using the same previous functional dependencies, a lossless-join decomposition and in BCNF? Explain your answer.

It is lossless join decomposition because: (ABCE)∩(BD)=B and B→D => B→BD
So the intersection is key to one of the relations.

It is in BCNF because:

- The only determinant is a key
- No transitive dependencies
- No dependencies on part of key

4- [3 points] BCNF is the desired normal form when designing a database. However, often, we may have to choose a non-BCNF. Why is that?

(1) Too many joins can outweigh the overhead of dealing with redundancy.

(2) Preservation of FD

5- [3 points] What are the options when a decomposition is non dependency preserving? Give a simple example.

(1)Create a relation that contains the attributes involved in the FD not preserved.

(2) Go back to a lower normal form

(3)Do a join for each insert to check dependency

Section 3: Multiple choice [30 points]

(3 points each for correct answer, -1 for wrong answer)
Circle exactly ONE choice as the best answer to each question.

1- Certain functional dependencies are called trivial because:
a) They have three attributes;
b) They are satisfied by all relations; ←←
c) They are not true functional dependencies;
d) They are too simple;
e) They are in first normal form.

2- A query tree:
a) Is a set of queries to be optimised;
b) May have different execution plans; ←←
c) Helps estimate the cost of a query;
d) Is an efficient index structure for queries;
e) Has indexed relations as leaf nodes.

3- The system catalog stores the index cardinality which is:
a) The number of distinct key values; ←←
b) The size of the largest bucket;
c) The number of pages containing the catalog;
d) The largest index value;
e) The number of indexes used in the database.

4- Indicate which statement is NOT true. When considering the creation of an index, the database designer should take into account:
a) The type of queries users usually submit;
b) The frequency of certain queries users submit;
c) The type of attribute comparisons in the WHERE clause of typical queries;
d) The existence of other indexes;
e) The size of the relation in number of disk pages. ←←

5- Which one is NOT a property of a transaction:
a) Atomicity;
b) Transitivity; ←←
c) Durability;
d) Consistency.

6- The two-phase locking protocol is called that way because:
a) There are 2 phases: a shrinking phase then a growing phase;
b) No locks can be requested until previous locks a released;
c) No locks can be requested once the first lock is released; ←←
d) Locks have two commands: lock and unlock;
e) Locking transactions either strictly lock or not other transactions;

f) Locking and unlocking are atomic operations.

7- Write-ahead logging protocol means:
a) Always keep on writing before commit;
b) Always write to the file before writing to log;
c) Always write to log before writing to file; ←←
d) Always check the file before writing to log;
e) Always write to log before checking file.

8- Checkpointing is:
a) A way to verify the consistency of the log;
b) A steal/no-force strategy for the log;
c) A no-steal/force strategy for the log;
d) A way to avoid backward scanning the whole log; ←←
e) A way to check the dirty pages at recovery time.

9- GRANT DELETE ON table TO user WITH GRANT OPTION:
a) Allows only user to delete table;
b) Is part of the mandatory access control mechanism;
c) Is part of the role-based access control for table;
d) Is an SQL command; ←←
e) Is used with statistical databases to delete table by user.

10- What is NOT written in the log:
a) Begin_transaction;
b) End_transaction;
c) Recover_transaction; ←←
d) Abort_transaction;
e) Commit_transaction.

Section 4: Query Optimization [35 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, Name, syllabus)
Teaching(P_ID, Code, Semester)

There are 1000 professors in 100 departments. The relation Professor is contained in 200 pages of disk, each page with 5 tuples of Professor. The Professor relation has 2 indexes: one hash index on P_ID and a clustered 2-level B^+tree on Dept_ID.

There are 2500 courses in the 100 departments. The relation is contained in 500 pages of disk, each page with 5 tuples of Course. The Course relation has 2 indexes: one hash index on code and a clustered 2-level B^+tree on Dept_ID.

There are 3000 teaching records for 3 semesters. The relation is contained in 300 pages, each page with 10 tuples of Teaching. The Teaching relation has 2 indexes: one hash index on P_ID and a clustered 2-level B^+tree on Semester.

1- [8 points] What are the relational algebra expressions that a query processor might generate given the following SQL query:

SELECT P.Name
FROM Professor P, Teaching T
WHERE P.P_ID = T.P_ID **AND**
T.Semester = "Fall 2001" **AND**
P.Dept_ID = "CMPUT"

a. $\Pi_{Name}(\sigma_{Dept="CMPUT" \wedge Semester="F2001"}(Professor \infty Teaching))$
b. $\Pi_{Name}(\sigma_{Dept="CMPUT"}(Professor) \infty \sigma_{Semester="F2001"}(Teaching))$
c. $\Pi_{Name}(\sigma_{Semester="F2001"}(\sigma_{Dept="CMPUT"}(Professor) \infty Teaching))$
d. $\Pi_{Name}(\sigma_{Dept="CMPUT"}(Professor \infty(\sigma_{Semester="F2001"}(Teaching))))$

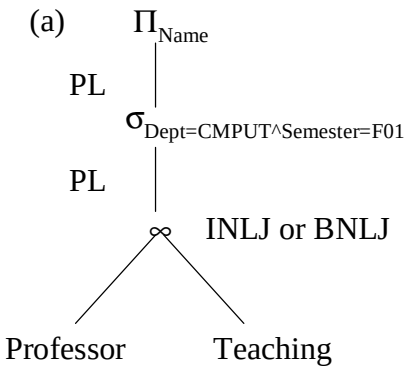
∞ is used as a symbol for join.

2- [8 points] Give the query execution plan for all the expressions in the form of trees knowing that we would always favour Indexed Nested Loop Joins if possible, otherwise we would use Bloc Nested Loop Join. Indicate which plan is for which expression (example: a-b-c-d):

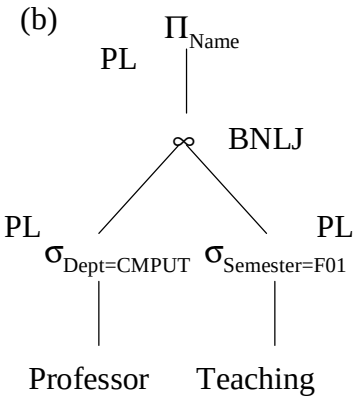
The query execution plans are as follows:

Note: PL means pipeline.

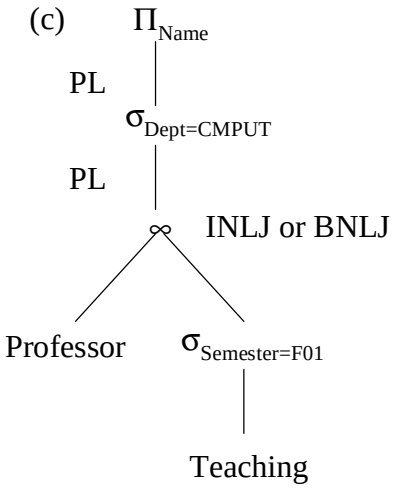
(a)



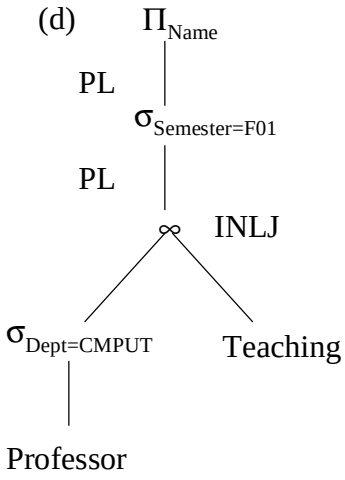
(b)



(c)



(d)



3- [16 points] Suppose we have 27 blocks in main memory, and assuming a uniform distribution for all the values in the database, estimate the cost in terms of I/O of the execution plans you provided in the previous question. Justify your answers. Indicate which cost calculation is for which execution plan (example: a-b-c-d).

(a) When using INLJ:

TO scan professor=200 pages
Use index nested loop join to join with Teaching
There are 3 teachings per professor
Search 1000 times at a cost of 1.2*1000 for the has index
Fetch teaching 3000 I/O
Hence it comes to 200+1200+3000=4400 I/O

When using BNLJ:

To scan professor =200 pages
Use BNLJ→25blocks→8 times (200/25) read blocks from Teaching
Fetch Teaching 8 times 300 pages=2400
Hence it comes to 200+2400=2600 I/O

(b) We use BNLJ.

1000 profs in 100 departments → 10 profs in CS→ 2 pages
To fetch the CS profs: 2 I/O for B+ Tree+ 2 I/O for pages=4 I/O since the professors are clustered by departments
3000 Teaching records in 3 semesters→ 1000 Teaching records in Fall 2001=100 pages since we have 10 tuples per page.
To fetch the Teaching records in F2001: 2 I/O for B+ Tree + 100 I/O since clustered by semester
25 blocks enough to hold 2 pages of profs → Total =4+102=106 I/O

(c) We use INLJ.

10 profs in CS→ 2 pages= 4 I/O with clustered B+ Tree
There are 3 Teaching per prof → 1.2*3=3.6 +3 (to fetch Teaching of each prof) and we have 10 profs.
Hence total cost=10*6.6+4= 66 +4=70 I/O

(d) 1000 Teaching records in F2001=100 pages=102 I/O with clustered B+Tree

Use teaching in outer loop to take advantage of index on profs, fetching pages =1000*1.2 +1000= 2200, hence we get to 2302 I/O if we use INLJ

If using BNLJ, 1000 teaching=100 pages→ 4 times (to read profs)→ 102 +4 *200=902 I/O

4- [3 points] Which execution plan should we choose for the query in question 1 and why?

Plan C should be chosen since its total cost is less than other plans.

(Push one selection and take advantage of hash index for inner relation)

Section 5: Concurrency Control [10 points]

1- [7 points] Consider the following transactions T_1 and T_2:

T_1:
Read(A)
Read(B)
if A==0 then B←B+1;
Write(B)

T_2:
Read(B)
Read(A)
If B==0 then A←A+1;
Write(A)

Add *lock* and *unlock* instructions to these transactions so that they observe the two-phase locking protocol.

T1:	T2:
Lock_S(A) Read(A) Lock_X(B) Read(B) If A= =0 then B←B+1 Write (B) Unlock(B) Unlock(A)	Lock_S(B) Read(B) Lock_X(A) Read(A) If B= =0 then A←A+1 Write (A) Unlock(A) Unlock(B)

2- [3 points] Can the execution of these transaction result in a deadlock? Give an example

Yes. Condider the following schedule:

T1	T2
Lock_S(A)	
	Lock_S(B)
	Read(B)
Read(A)	
Lock_X(B)	
Wait	Lock_X(A)
	Wait