Question 1. [4 MARKS]

Memory Hierarchy The data transfer speed (from memory to hard drive or vice versa) depends on (circle all that are correct).

1. how records are laid out within a page
2. contiguousness of the accessed physical locations on the disk
3. buffer replacement policy
4. the size of the accessed data per request

Solution: FTFT

Question 2. [8 MARKS]

File Organization Consider three separate file organizations for storing a relation $R[A, B]$ with primary key A: 1) heap file using two pages to store the page directory, 2) B$^+$tree index with search key A with height 3, or 3) extendible hash index with search key A and a one page directory. Both indices are unclustered and store data entries using Alternative 2 (meaning a data entry is a key value, record id pair).

A cost of O(1) means on average a low number of I/0 (say $<$ 5-6 pages) with the current file size. Which of these structures support the following (circle each correct answer).

1. Random access (direct access) by record id.
   Heap B$^+$tree Extendible Hash
2. Cost of O(1) for the insert: `insert into R values (A=1,B=2)`.
   Heap B$^+$tree Extendible Hash
3. Cost of O(1) for the update: `update R set B = 3 where A = 2`.
   Heap B$^+$tree Extendible Hash
4. Cost of O(1) for the deletion: `delete from R where B = 3`.
   Heap B$^+$tree Extendible Hash

Solution:

1) none (FFF)

2) all (TTT)

3) tree and hash (FTT)

4) none (FFF)

2 points each if all correct. 1 point if one is wrong (each). 0 points if two or three are wrong.

Name: _____

Question 3. [7 MARKS]

Query Sizes In this question, please estimate the query size in **tuples** (not number of pages). Suppose we have the following relations and statistics about these relations. Supplies.sno is a foreign key for Supplier and Supplies.pno is a foreign key for Part. There are 100K (meaning 100,000 or 10^5) Part tuples. There are 10K distinct part names and 1K distinct cities in Part. There are 100K Supplier tuples. There are 50K distinct supplier names and 1K distinct cities in Supplier. There are 1B (meaning 10^9) Supplies tuples. No other statistics are available.

`Part[pno,name,city]  Supplier[sno,name,city]  Supplies[sno,pno,proj,quantity ]`

Part (a) [2 MARKS]

Using these statistics, what will the optimizer estimate is the size (number of tuples) of this query?

select s.name
from supplier s, part p, supplies u
where s.sno = u.sno and p.pno = u.pno and p.name = "hammer"

Solutions: $\sigma_{name}(Part)$ gives about 10 parts, i.e., number tuples of part (10^5) divided by number distinct names (10^4).

There are 10,000 Supplies per part (i.e., #Tuples in Supplies/ #Tuples in Part = $10^9/10^5$=10^4).

$\sigma(Part) \bowtie U$: So join of Part and Supplies gives about 100,000 result tuples (10,000 times 10).

$(\sigma(Part) \bowtie U) \bowtie S$: For each supplies tuple there is exactly one supplier so this join does not change the cardinality.

Estimate: 100,000 = 10^5 tuples

Part (b) [2 MARKS]

Assuming the statistics are correct, what is the lowest and highest number of tuples the query could actually return? (That is, how far off could this estimate be?)

Lowest: 0 (for example if there are no hammers in the db)

Highest: there could be up to 90K hammers and it could be that only hammers are supplied, so all 1B Supplies tuples could represent hammers. Hence, the max is 1B (10^9) tuples.

Note that the joining 1B Supplies tuples with Part and Supplier can never result in more than 1B tuples (because of the key constraints on Part and Supplier).

Part (c) [3 MARKS]

If the `select` clause is changed to `select distinct`, how will the estimate of the query size change? Again, assuming the stats are correct, how far off could this estimate be (what is the minimum and maximum number of tuples this query could return)?

select DISTINCT s.name
from supplier s, part p, supplies u
where s.sno = u.sno and p.pno = u.pno and p.name = "hammer"

Estimate: 50K tuples (number of distinct names). To arrive at this, consider that without the distinct we are estimating 100,000 tuples in $\sigma(Part) \bowtie U$. Assuming uniform distribution, the optimizer would assume this is joining with 100,000 distinction Suppliers (i.e., all Suppliers) and we know that there are 50K distinct names among these.

Min: 0 (if there are no hammers)

Max: 50K (can't be higher is stats are correct since there are only 40K s.names in the db).

Question 4. [16 MARKS]

Query Plan Cost Estimation Consider the following relations where the key of the relation is underlined, all attributes are declared to be non-null, and the statistics known to the optimizer are given below. There is a clustered B^+-tree index on Employee.Sal, an unclustered Hash index on Emp.Name and another on Emp.Dept. There is a clustered B^+tree on Dept.Name. The B^+-trees have a height of two (meaning two index pages, this does not include the data entry level of the tree). Assume that for the Hash index, the cost of a probe is 1 data-entry page plus the cost to retrieve the selected tuples. For this question, consider sort-merge (SM), hash (HJ), block nested loop (BNL), and index nested loop (INL) joins only. Assume there are twenty buffer pages available for each operator.

`Emp[Name, Proj, Sal, Dept], Dept[Name, HQ, Head], Project[Name, Budget]`

`Pages(Emp)=1,000 Tuples(Emp)=10,000`
`Number Distinct Values of Emp.Proj (i.e.,` $|Emp.Proj|$`) is 100`
`Number of Distinct Values of Emp.Sal=5,000, lowest=1, highest 100,000`
`Number of distinct values of Emp.Dept=1,000`
`Pages(Dept)=100 Tuples(Dept)=1,000` $|Dept.HQ| = 1,000 |Dept.Head| = 50$
`Pages(Project)=100 Tuples(Project)=1,000` $|Project.Budget| = 1,000$

SELECT *
FROM Emp E, Dept D, Project P
WHERE E.Dept = D.Name and E.Proj = P.Name and E.Sal > 99,000

In the first pass, the optimizer will consider all one-relation plans. There are five "winning" plans that are used in the second pass.

For Project, there is only one access method, a filesscan. This is selected as a winner. Call this plan fs(P).
For Dept, a filescan is the lowest cost access method so it is kept as a winner. Call this plan fs(D). However the index on Dept.Name is an interesting order so it is also kept as a winner. Call this plan index(Dept.Name).
For Emp, the index selection using the Sal index that evaluates the selection condition is a lower cost plan than a filescan, hence index($\sigma_{Sal}(Emp)$) is kept as a winner. Finally, Emp.Dept is an interesting order so the index on Emp.Dept is also kept as a winner. Call this plan index(Emp.Dept).

Part (a) [2 MARKS]

What is the size of the plan index($\sigma_{Sal}(Emp)$) (meaning the number of result tuples)? What is its I/O cost?

Solution: size = 100 tuples (we are selecting 1/100th of the tuples and there are 10,000 tuples).

I/0 we are using a clustered B+tree with height 2, one data entry page (assume 100 data entries fit on one page), and then retrieving 100 tuples which are stored clustered and 10 tuples per page. Hence, we estimate about 10 pages of tuples are retrieved.

Note: I did not specify the number of data entries per pages, so if they assumed fewer data entries per page (like 10), then that would mean more data entires (for 10 entries per page, we have 10 data entry pages).

Total: 2 (index pages) + 1 (data entry page) + 10 (relation pages) = 13 pages.

Part (b) [14 MARKS]

Indicate all 2-relation plans considered by the query optimizer. Indicate their cost and which plans would be selected as "winners" to be considered when constructing three relation plans. Clearly indicate the reason for this selection. For each plan, indicate the size of the result.

Reminder: for each Plan i: a) describe plan b) give the I/O cost of the plan c) indicate the size of the plan output d) indicate if the plan is a "winner"

Hint: you will use all five of the one-relation plans, at least once.

Solutions: be lenient with small math errors, the goal is for them to understand the general trends. So more points off for including something that doesn't make sense (a plan I don't have) than for missing one or two of these).

Plan$_{ED}$ 1: ED: index(sigma(E) HJ D

sigma(E) produces 100 tuples which fit in 10 pages (and therefore in memory) so we can hash them in memory, read D one page at a time and hash each tuple to get join result.

I/O cost is cost (index(sigma(E)) = 13 plus Pages(D) = 100 Total: 113

Plan$_{ED}$ 2: ED: index(sigma(E) SM D - can sort E in memory and don't need to sort dept since there is clustered B+ tree index. So I/O is same as above. -2 if they didn't realize D is sorted.

I/O cost is cost (index(sigma(E)) = 13 plus Pages(D) = 100 Total: 113

Plan$_{ED}$ 3: ED: index(sigma(E) BNL D again since E fits in memory cost is just to read D. Total 113 as above

Plan$_{ED}$ 4: DE: D BNL index (sigma(E)) here there would be 5 or 6 blocks (i.e., 100/(20-2)) of D so cost is:

Pages D + blocks D (pages(sigma(E)) = 100 + 6(10) = 160.

Plan$_{ED}$ 5: ED: index(sigma(E)) INL D for each tuple E probe index. Probe cost is 2 +1 + 1 = 4.

Worst cast i/O cost is cost (index(sigma(E)) = 13 plus + 100(4) = 413
(Note you can do better with a good buffer replacement policy, so please allow for that in their answers.)

Plan$_{ED}$ 6: ED: D INL index on dept E (note that selection is not pushed so that you can use index on E). Cost is Pages (D) + tuples(D)(probe cost on E.Dept). Probe cost is one data entry page plus (estimated) 10 Emp per dept. Index is unclustered so these can be on 10 different pages.

100 + 1,000*(11) = 11,100

Size Plan$_{ED}$s 1-5: selection is push so must be part of the plan. The selection retries 100 Emp tuples and each has exactly one dept so result size of the plan is 100 tuples.

Size Plan$_{ED}$ 6: if you don't push selection the join would have 10,000 tuples in it. If you pipeline the selection after the join, then you'd get 100 tuples (as in Plan$_{ED}$s 1-5). They may give either answer but sound have observed that if the plans have the same operators in them they should be computing the same result and hence it should have the same size! -3 at least if they missed this Plan$_{ED}$ 6.

So winner is any of the plans with cost 113 (Plan$_{ED}$ 1, 2 or 3) - so the INL is not helping as much as pushing the selection.

Note: the qoptimizer would not consider other plans like D sort-merge E without pushing the selection (that is the beauty of the dynamic programming pruning).

For EP only consider result of sigma(E) since index on Emp.Dept is not an interesting order for this join. -3 if they considered this index for EP join.

Plan$_{EP}$ 1: (index(sigma(E)) HJ P again E tuples fit in memory 10 pages so cost is just

I/O cost is cost (index(sigma(E)) = 13 plus Pages(P) = 100 Total: 113

Plan$_{EP}$ 2: (index(sigma(E)) SM P E can be sorted in memory but need an external sort for P. One read/write pass (200 pages) creates 5 (or fewer if you use optimizations discussed in class)

runs. So you can do join in next read (100 pages) by just reading one page of each of the 5 runs at once. -2 if they assumed E was sorted or got cost wrong.

I/O cost is cost (index(sigma(E)) = 13 plus Sort(P) = 3(100) = 313.

Plan$_{EP}$ 3: (index(sigma(E)) BNL P - E fits in memory so you just need one pass of P.

I/O cost is cost (index(sigma(E)) = 13 plus Pages(P) = 113.

Plan$_{EP}$ 4: P BNL (index(sigma(E)) - (same as D BNL sigma(E) - Plan$_{DE}$ 4) there would be 5 or 6 blocks (i.e., 100/(20-2)) of P so cost is:

Pages P + blocks P (pages(sigma(E)) = 100 + 6(10) = 160.

So winner is any of Plan$_{EP}$ 1-3.

Don't consider any INL for EP. -3 if they got this wrong.

Size is 100 tuples (each of the 100 Emps has exactly one project).

Don't consider P⋈D since this is a cross-product

Question 5. [12 MARKS]

Consider the following classes of schedules: serial, view-serializable (VS) conflict-serializable (CS), recoverable, no-cascading-aborts, 2PL (two-phase locking) and strict 2PL. For each of the following schedules, state which of the above classes it belongs to - indicate **T** next to the class if the schedule is in the class and **F** next to the class if the schedule is not in the class.

The actions are listed in the order they are scheduled, and subscripted with the transaction number. For simplicity, we assume the listed transactions are the only ones active currently in the database.

Marking: 3 points if all answers are correct; 2 points if one answer is wrong; 1 point if two answers are wrong. No points if three or more answers are incorrectly marked.

Part (a) [3 MARKS]

$R_1(X), R_1(Y), W_1(Y), Abort_1, R_2(Y), Commit_2$

Solution: TTTTTTT

Part (b) [3 MARKS]

$W_1(X), R_1(X), R_2(Y), W_1(Y), W_2(X), W_1(X), Commit_2, Commit_1$

Solution: FTFTTFF not serial is VS, not CS is recoverable and ACA not 2PL; is strict 2PL.

Part (c) [3 MARKS]

$W_1(Y), R_1(Y), R_2(Y), W_2(X), Commit_2, Commit_1$

Solution: FTTFTF Not Serial is VS and CS not recoverable has cascading aborts is 2PL not strict

Part (d) [3 MARKS]

$R_1(X), W_2(X), W_1(X), R_3(X), Commit_1, Abort_2, Commit_3$

Solution: FFFTFFF not view/conflict-serializable. It is recoverable, but does not avoid cascading aborts. Not 2PL or strict.

Question 6. [8 MARKS]

Recovery Assume we use Aries write-ahead logging for recovery (so we are using a no-force policy and allowing steals). Consider the following sequence of undo/redo log records. Each log record includes (in order) the LSN, the transaction id, the log record type. The last entry is the previous LSN. For updates, the log record also includes the object id (P_i), the before-image (former value) of the object and the after-image (new value) of the object. Compensation log records contain the same information as update records with one final value which is the undonextLSN (the next LSN that needs to be undone). For brevity, we don't include the length and offset of the change. The transaction ids are T_i. For this question we assume page-level logging so the object ids are page ids P_i.

Assume the log starts at LSN 1, but there are no actions by T_1, T_2 or T_3 before LSN 100. There are no other transactions that are active at the time of the crash which happens after all log records below are written to disk. There may be other log records (with higher LSNs that are in memory at the time of the crash).

....
[100,T_1,UPDATE,P_1,1,2,$\emptyset$]
[101,T_1,COMMIT,100]
[102,T_3,UPDATE,,P_2,1,2,$\emptyset$]
[103,STARTCHECKPOINT]
[104,T_3,UPDATE, P_3,1,2,102]
[105,T_1,END,101]
[106,ENDCHECKPOINT,DPT(P_1,100), XT((T_3,102, running),(T_1,101,commit))]
[107,T_2,UPDATE,P_1,2,3,$\emptyset$]
[108,T_2,UPDATE,P_3,2,3,107]
[109, T_3,Abort,104]
[110,T_3,CLR, P_3, 2,1,109,102]
[111,STARTCHECKPOINT]

Part (a) [4 MARKS]

Give the dirty-page-table and transaction table after the end of the analysis phase.

DTP: (P1,100) (P3,104)

XT: (T3,110, abort), (T2,108,running)

(ok if they include (T1, 105, end/commit) but really this should be removed from XT).

Part (b) [4 MARKS]

List, in order, the undo actions that must be done for recovery. Be specific by also giving all the CLR log records written as part of undo.

Undo 108: bring P3 into memory and if pageLSN = 108 set value from 3 back to 2.

Undo 107: bring P1 into memory and if pageLSN = 107 set value from 3 back to 2.

Undo 102: bring P2 into memory and if pageLSN = 102 set value from 2 back to 1.

[112,T2, CLR, P3, 3, 2, 108,107]

[113,T2, CLR,P1, 3, 2, 112, $\emptyset$]

[114,T3,CLR, P2, 2, 1, 110, $\emptyset$]

OK if they have three CLR records exactly right, otherwise, need to look at explaination of what they are undoing.