CSC 321/621 Spring 2012
Wake Forest University
Final Exam – May 10, 2012

Name: KEY

Show all your work and reasoning for your answers in the space provided. Only one solution per question will be accepted, so be certain you clearly mark your final answer if you have a lot of scratch work for a problem. This exam is to be done individually according to the honor code at Wake Forest University.

Problem 1 Transactions & Recovery

a. Indicate, with one word each, what term the 'I' and 'D' in the ACID properties of transactions stand for and then, in a few sentences, indicate what those terms mean.

I - isolation - transactions should occur independently of each other. Partial effects should not be visible to other transactions.

* looked for both of these

D - durability - a commited transaction should be considered as permanently recorded in the database and should be maintained (recoverable) even after failure

* looked for these

b. Briefly describe what 'recovery protocols' are (in general, not specific to any particular recovery protocol we studied) and how recovery protocols enable the 'D' property of transactions.

Processes or algorithms that ensure the four properties of transactions when faced w/ database failure. Typically determine when to undo or redo changes to a database depending on the outcomes of the transactions (abort/commit) and the state of the database at failure time.

* key ideas

Problem 2 Concurrency control

Consider the following history of transaction actions on data. Time flows downwards (T1 is earlier than T3).

Transactions →

| Time/Transaction | TransactionA | TransactionB |
|---|---|---|
| T1 | Read(DataX) | |
| T2 | | Write(DataY) |
| T3 | Read(DataY) | |

Time ↓

a. Is this history possible if a database is using a 2-phase locking (2PL) mechanism? Indicate YES or NO and argue why or why not. Locking actions are not explicitly represented here, but you can assume the appropriate actions happen as late as possible for requests and as soon as possible for releases, all still premised on the notion that everyone is working under the the 2PL locking protocol.

2PL - request when needed
release when done w/ all locks

yes, it is possible.

A requests shared lock on Data X, it is granted.

B request exclusive lock on Data y, it is granted.

B is done at that point, releases lock on Data y.

A requests shared lock on Data y, it is granted.

* Note Time ($T_1, T_2, T_3$); Transactions A,B

b. Regardless of your answer to Part (a) above, what equivalent serial schedule does this history represent and why?

Schedule over transactions

so either $T_A, T_B$ or $T_B, T_A$

Transaction B, Transaction A

Equivalent to the following: B writes y, A reads x, A reads y which is serial (no interleaving)

but only $T_B, T_A$ matches

Problem 3 Fragmentation

Assume the following is an instance of a Hotel relation in a HotelManagement database. (It is small to make this problem easier!)

**HOTEL Table**

| HotelID | HotelName | HotelChain | City | OccupancyLimit |
|---|---|---|---|---|
| 00001 | University Quarters | Marriot | Winston-Salem | 100 |
| 00002 | Twin City Towers | Doubletree | Winston-Salem | 600 |
| 00003 | Capital Hotel | Marriot | Raleigh | 500 |
| 00004 | Tarheel Tower | Marriot | Raleigh | 350 |

a. Fragment this relation into 2 or more fragments– how you fragment it is up to you, but I would argue you should choose a reasonable fragmentation because you will have to justify it later. Provide appropriate selection or projection statements here that define your fragmentation (at least 2; your SQL doesn't have to be perfect, but get the idea of how you are fragmenting across):

One possible example: (assume table represents all data)

2 queries defining fragments:

① Select * from Hotel where HotelChain = "Marriot"

② select * from Hotel where HotelChain = "Doubletree"

Dividing on City would also have been appropriate.

b. What is the purpose of your fragmentation (why is it reasonable/potentially useful)?

For my example an argument is:

Dividing data by company chain would allow each chain to a) manage/maintain their own chunk of data and b) have fastest access to such data. A chain is an organizational unit and may have particular data policies unique from other chains and will likely use the data about reservations at hotels in their chain most often.

* looking for a 'role' or 'physical' location argument

c. Choose 2 of the 3 properties a valid fragmentation must exhibit, name and define those properties, and demonstrate your fragmentation meets those properties.

Complete: All orginal data (each tuple) must appear in some fragment.

→ Example: There are only two chains in the table. My fragmentation queries cover both chains, so no data has been looked over / forgotten about.

* Needed to give a statement providing proof → something like "my two queries, one using =, one using != , cover all cases"

Disjoint: No data (tuple) appears in > 1 fragment.

Assuming hotels don't belong to 2 chains (they don't in real life), each tuple will only be selected by 1 or the other query, but not both.

Reconstructable: Via union (3 Marriot hotels (fragment 1) ∪ 1 Doubletree hotel (fragment 2) ⇒ all 4 hotels)

* Needed to give an appropriate relational operator that reconstructs

* - use view materialization when not many changes as creates a table that is stored
- use resolution if underlying data is changing a lot, compute on the fly

Problem 4 Views

a. Below I will provide four views defined on a large student database. Beside each view, indicate whether you would use "view materialization" or "view resolution" in implementing the view AND support your choice for each one.

*A view that provides Deacon Card Balances (how much money is left in their dining account):*

The balance probably changes daily (maybe > 1 time a day)
I would use view resolution

*A view that provides Student Contact Information (mailbox, phone number,...)*

This rarely changes in my opinion (once a year?, if move rooms in a residence hall)
I would use materialization.

*A view that provides Student GPA Information:*

This, I expect, only gets updated at the end of each semester.
I would use materialization.

*A view that provides Student Attendance Records (whether they come to each class or not):*

This is updated every day (every day of class) for lots of students, and probably > 1 time a day since students often have 2 to 3 classes a day.
I would use resolution.

b. Under what scenarios could a table that a view is defined on be modified but the view not be modified?

If the view doesn't involve the modified attributes.
(As an example, The attendance records view would not be modified if the Student GPA changes!!)

Problem 5 Distributed Transaction Recovery

a. What are the 2 phases of the 2-phase-commit protocol? (just give a name for each and a short summary of what is happening in each phase)

Voting: Participants are indicating whether they are / need to abort their subtransaction or whether they are prepared to commit it.

Decision: Participants are actually implementing the decision made/ agreed upon in the voting phase.

b. Why is a commit protocol even needed with transactions executing within a distributed database?

A transaction in a distributed database is formed out to individual databases as sub-transactions. The overarching transaction must be thought of / treated as if it is a single entity. Thus all subtransaction must commit or all subtransactions must abort, not some mix.

* key ideas

The commit protocols support gathering information from and acting at the Participant level in handling the overarching transaction.

c. Argue FOR or AGAINST the following statement AND justify your answer: *It is possible to see a unilateral (single computer) abort in the second phase of the 2-phase-commit protocol.*

AGAINST.

* looked for stmts that "all commit or all abort" or "all do same thing"

At the decision phase (phase 2), everyone is commiting (so no-one aborts) or everyone is aborting (so all abort). You should never see one abort by itself (unilaterally) at that point.

Everyone has agreed to do one or the other and must follow through.

Problem 6 Logging and Recovery

Below is a set of transaction start and completion times and a record of what happened at completion (commit or abort (abort is the same as 'rollback')). If a time has the word 'Expected' next to it in the table, it is the time the transaction would have finished if failure would not have occurred. Separately are listed two checkpoint times and the time of a failure. You are using **'Immediate Update'**.

***Checkpoint times: 15, 30***
***Failure time: 42***
***Transaction information:***

| Transaction | Start Time | Complete Time | Completion Action |
|---|---|---|---|
| T1 | 2 | 12 | Abort (Rollback) |
| T2 | 4 | 20 | Commit |
| T3 | 5 | 14 | Commit |
| T4 | 12 | 29 | Commit |
| T5 | 18 | 35 | Commit |
| T6 | 29 | 44 (Expected) | Commit |
| T7 | 35 | 50 (Expected) | Commit |

a. Indicate which, if any, transactions have their data all written to disk at the time of the first checkpoint and how you know that.

T3 - commits before first checkpoint
at time 14 at time 15

Note - T1 itself aborted by choice - not written to disk ever during a checkpoint

b. Indicate which, if any, transactions have their data all written to disk at the time of the second checkpoint. Here you can just list any such transactions.

(T3 was already written)

T2 and T4 both commit before the second checkpoint

c. Indicate for all transactions, what action is required (your choices: *do nothing, redo, or undo & restart*) after recovery from the failure AND why each such action is required.

T1 - do nothing - it aborted itself and was never written to disk

T2, T3, T4 - do nothing - these have committed and been written to disk at checkpoints 1 and 2

T5 - redo - committed before failure but after checkpoints

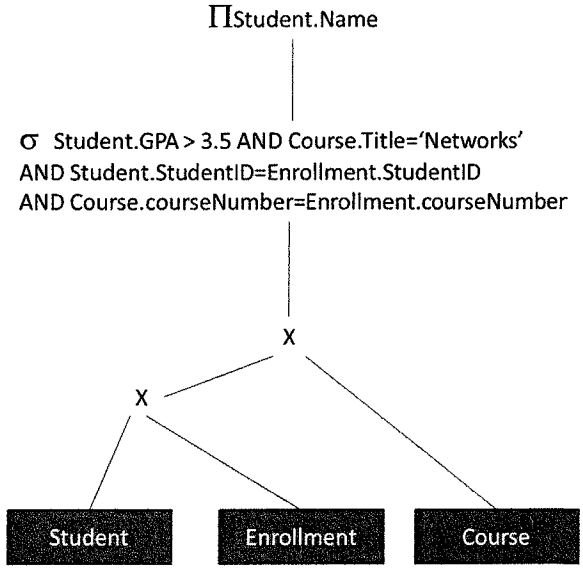
T6, T7 - undo + restart - active at failure

*Redo is to redo a committed transaction; undo + restart is to back off partial changes from uncommitted

Problem 7 Query Optimization

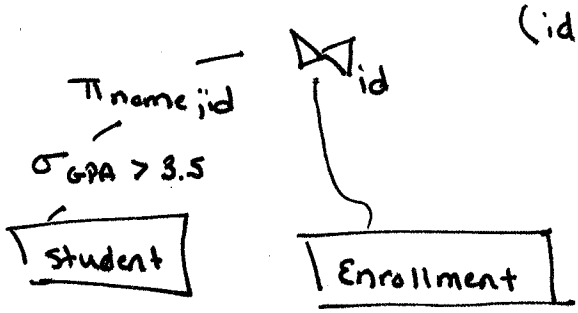
a. Given the tree below, what query does the tree represent? Please answer in SQL terms or in "plain English" terms.

Show me all students enrolled in the 'Networks' course with GPAs > 3.5




b. Suggest two improvements to the tree that use different rules for improvement. Relational algebra transforms that can be applied are available on the last few pages of the test, as is a page containing which relational algebra operators mean what. Be detailed enough in your descriptions of what improvements you are making that it is very clear to the reader! (Alternatively, redraw enough of the tree with your improvements in place to be clear).

The following is an example of three improvements to the bottom left of the tree.




* Note - just nesting selects isn't really an optimization; nesting (to split apart) and then pushing down to Relations is!

c. For one of your improvements, argue why it is legal and argue why it is an improvement.

* was looking for reference to one of rules in legality claim

Pushing $\sigma_{GPA>3.5}$ to just above Student is legal as:

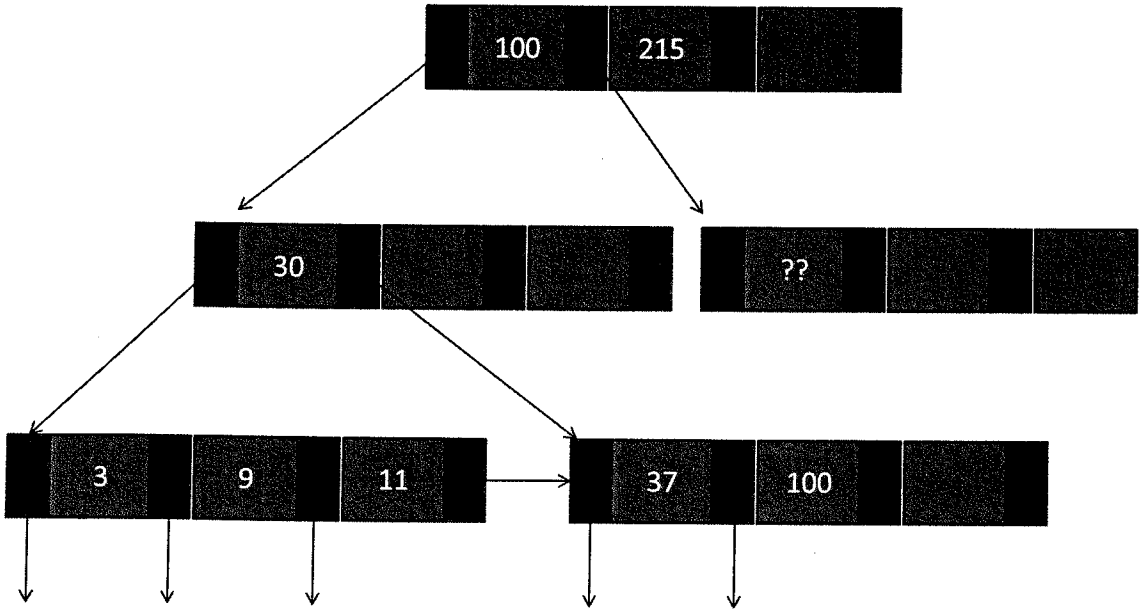
① an AND'ED select can be broken into nested selects ↔ $\sigma_{p\wedge q}(R) = \sigma_p(\sigma_q(R))$

② the select only involves attributes of Students relation ↔ using $\sigma_p(R \bowtie S) = (\sigma_p(R)) \bowtie S$

Improvement: original tree joined all Students w/ all enrollment data, basically generating everyone's schedule of classes. This choosing a likely much smaller set of students as inputs into that join

Problem 8 Indexing

Below is part of (most of) a B+-tree-index for some relation. It is of height 3 and each node holds 3 pieces of data (light gray color) and 4 pointers (black color). Assume it has been validly constructed to this point. Any missing nodes/arrows are irrelevant for the problem.




a. Provide a value that could be in the slot labeled '??' and indicate why that value would be correct sitting in that spot.

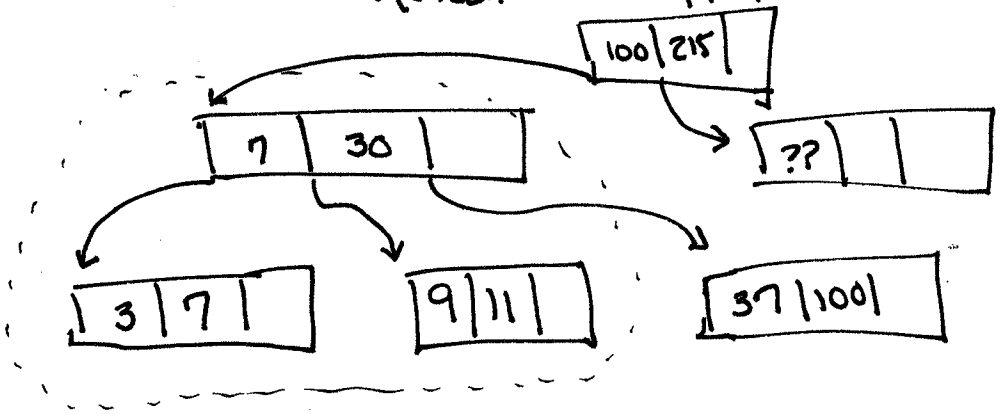
108 is one example

must be > 100, ≤ 215 by rules of how B+ tree indices are searched

* looked for both a has to be < X and a >y statement

b. Draw a revised picture of the B+-tree-index after the value '7' is inserted into the tree. You should redraw all current nodes and any new nodes required.

Inserting '7' forces a split of the 3, 9, 11 node. One value from the split is copied into parent. The parent has open spaces to absorb the copy, so no other work is required. An appropriate model after insertion:



c. Assume the tree is of size *h* and each node has at most *k* entries (for this tree h = 3, k = 3). One typically measures the costs of search in these tree as O(h), since we are interested in number of pages visited. If we were working at a "number of operations level", make an argument that one might claim the cost of working with the tree is O(h*k). [Rephrased, this question is asking: what is the worst case scenario for using a B+-index tree, and why does it require h*k amount of work?]

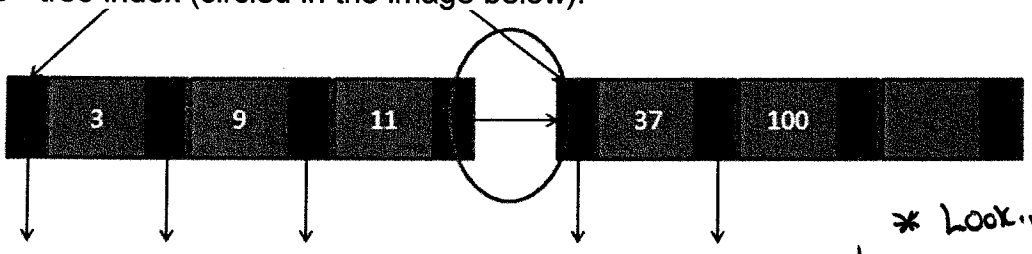
always visit one node per level ⇝ h nodes

In a node, may have to compare value searching for against all entries = K values per node

⇝ h nodes · $\frac{K \text{ values}}{\text{node}}$ ≈ h · K values compared against ("operations")

* several people rightly claimed (I wasn't looking for this, but it's interesting) - this only happens w/ a full tree, and searching for node in bottom right
* note, question only asked about search costs, not insert costs

d. Briefly describe the purpose of the horizontal pointers that appear only on the last level of the B+-tree index (circled in the image below).




* Looking for more than "just links together"

It allows for fast sequential searches / range searches

what was it's utility/benefit?

Graduate Student Problem 1 Views, Query Optimization

Views are defined via queries on database tables and provide us with what is essentially a 'virtual table' which we can define further queries on (queries on the views themselves). That means it seems reasonable to provide the capability to optimize queries on views much like we optimize queries that we allow directly on the raw database tables.

As you know, views can be implemented by view materialization and view resolution.

a. Argue why query optimization on views implemented using view materialization is likely relatively ineffective compared to query optimizations on views using view resolution.

when optimizing a query, we would ideally like to be given the entire relational algebra tree version of the query and be able to transform it. The problem w/ view materialization is we get stuck having to work w/ an intermediate table pre-built for us and have to work w/ that. If we deal w/ view resolution, where we essentially build a new query, we can likely highly optimize the query and may not have to generate a table anything like or near the size of the table made as the temporary table for view materialization.

* key idea looking for

b. Database Dan has just published a paper arguing for a "view optimization cache", a place to store transformed relational algebra trees so they can be looked up quickly. Indicate whether you think this is a useful idea or not in terms of optimizations of queries on views that are implemented via view resolution. Be sure to justify your choice of whether it is useful or not.

* I was generous but was looking for notion - some large chunk of tree optimization in this situation is the same every time

I would argue this is a good idea. Each query on the view will turn into a modified query on the underlying table. Note, however, that they (all the queries) will be employing a set of the same operations (those that came from defining the view) and a set of new operations, different per query (from the query on the view). Why not store off the tree optimized for the predicates defining the view (since for any query on the view you will have to repeat that), load it (partially optimized) when needed, and make the optimizations specific to the query on the view?

(probably need some things that can't get to optimal level tree from modifications to a partially optimized tree)

Graduate Student Problem 2 Triggers

You should be aware at this point in the course that any reasonable database implementation should support the notion of *referential integrity*. Usually a DBMS will support behind the scenes management of *"on delete" and "on update" cascade and restrict constraints*. These respectively propagate or prevent changes to database tables to preserve foreign key relationships. Here's a claim that I make:

***Triggers** are a mechanism that could be used to implement the needed referential integrity actions to support "on delete" and "on update" cascade and restrict constraints.*

Give a justification of WHY triggers should work in this capacity and discuss how one would set triggers up to use them in this capacity.

Triggers allow an event/code to run before or after a database insert/delete/change.

delete and update cascades could be implemented as after triggers - when a change is made to one relation, propegate its effects to other relations

restricts could be implemented as before triggers - don't allow a change to a relation if it violates a rule

* Need to discuss:
  - triggers (before + after firing) in general
  - cascades ('after' triggers)
  - restricts ('before' triggers)