My Name: _____SOLUTIONS_____

My Course Account: _____

Midterm 2: CS186, Spring 2015

Prof. J. Hellerstein

You should receive a double-sided answer sheet and an 8-page exam.

Mark your name and login on both sides of the answer sheet, and in the blanks above.

For each question, **place only your final answer on the answer sheet**—do not show work or formulas there.

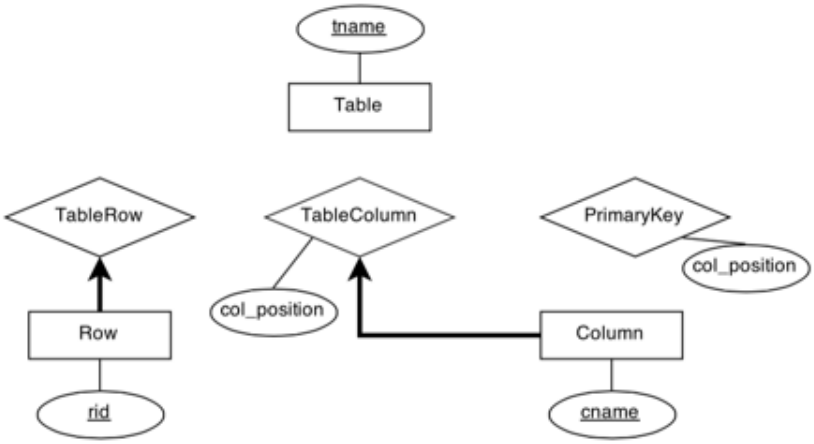
You may use the backs of the questions for scratch paper, but do not tear off any pages.

We will ask you to turn in your question sheets as well as your answers.

I. ER Diagrams [18 points]

1. [4 points] **True or False:**
   a. ER diagrams are used to model the physical schema of a database.
      False, they are used to model the logical schema. Physical schema refers to things like file and index layout.
   b. An entity set E can be associated with a relationship set R more than once.
      True, for example a relationship Reports_To which has the entity Employees participating twice, since Managers are also Employees.
   c. At most 2 distinct entity sets can be associated with any relationship set R.
      False, for example, a ternary relationship "Covers" with participating entities Employees, Dependents, and Policies.
   d. A weak entity set must have total participation in its identifying relationship set.
      True, by definition.

Chancellor Dirks wants you to explain how a **relational database management system**, (e.g. Postgres) works. Dirks is familiar with reading ER diagrams, so we've created the diagram below, but it is missing a few edges.



Answer questions 2 to 5 choosing the best option from the following multiple choice options:

A. partial participation, non-key
B. partial participation, key
C. total participation, non-key
D. total participation, key
E. none of the above

2. [2 point] What is the correct edge between TableRow and Table?
A - a table has 0 or more rows
3. [2 point] What is the correct edge between TableColumn and Table?
C - a table has 1 or more columns
4. [2 point] What is the correct edge between Column and PrimaryKey?
B - Each column participates in at most one primary key. There is at most one primary key, there may be none, and even if there is one not every column will be in it.
5. [2 point] What is the correct edge between Table and PrimaryKey?
A - A table may have a primary key with 0 columns (meaning it doesn't have a primary key), 1 column, or many columns, so the table can participate in the PrimaryKey relationship 0 or more times.

ER to SQL
6. [6 points] On your answer sheet, fill in the blanks to convert the **Row entity set**, and the **TableColumn relationship set** in the ER diagram above into SQL tables. You may not need all the blanks.

```
CREATE TABLE Row (
  rid INTEGER
  PRIMARY KEY (rid)
);

CREATE TABLE TableColumn (
  tname STRING,
  cname STRING,
  col_position INTEGER,

  FOREIGN KEY (tname)
      REFERENCES Table
  FOREIGN KEY (cname)
      REFERENCES Column
  PRIMARY KEY (tname, cname)
);
```

Our intended answer for the PRIMARY KEY of Table Column was (tname, cname) since we usually expect a column to be identified by the table it belongs to, e.g. Students.id.

However, since cname was shown to be a key for the Column entity, meaning that all column names are unique in this model, we also accepted `cname` and `(cname, col_position)` as answers for the PRIMARY KEYs.

To make our ER diagram more accurate for Chancellor Dirks to understand, a better choice would be to make Column a weak-entity of Table. It's a good exercise to think about what this would look like in the diagram and in SQL.

II. FDs & Normalization [16 points]

1. [4 points] Consider the attribute set R = ABCDEF and the functional dependency set F = {AD → B, A → E, C → E, DEF → A, F → D}. Find a candidate key of R.
   FC
   Notice that FC is not on the right-hand side of any of the given FDs. This means that any key of R *must* contain FC. Find the closure of FC to see that FC -> R. Thus, FC is a candidate key. Any other key would be superkey, since a key must contain FC.

2. [6 points] Given the attribute set R = ABCDEFGH and the functional dependency set F = {BC → GH, AD → E, A → H, E → BCF, G → H}, decompose R into BCNF by decomposing *in the order of the given functional dependencies*.
   ADE, BCEF, GH, BCG
   BC -> GH violates BCNF, decompose.
   Relations: ABCDEF BCGH
   AD -> E does not violate BCNF because AD is a superkey, skip.
   A -> H No relation contains AH, skip.
   E -> BCF violates BCNF, decompose.
   Relations: ADE EBCF BCGH
   G -> H violates BCNF, decompose.
   ADE EBCF BCG GH

3. [6 points] Given the attribute set R = ABCDEF and the functional dependency set F = {B → D, E → F, D → E, D → B, F → BD}.
   a. Is the decomposition ABDE, BCDF lossless?
   b. If not, what functional dependency could you add to make it lossless?

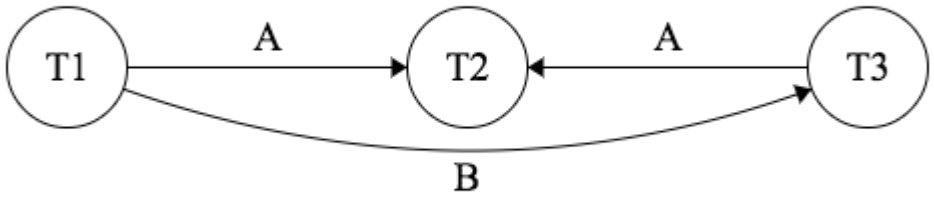
   a. **No**, it is lossy
   ABDE ∩ BCDF = BD
   BD -> BDEF, which is *not* equivalent to either ABDE or BCDF
   b. **Any or all of BDEF -> Any or all of AC**
   For example some valid answers were: B -> C, B -> A, BEFD -> AC, etc.
   To be lossless, we want BD -> ABDE or BD -> BCDF. We know that BD -> BDEF, so either want BD -> ABDEF or BD -> CBDEF (or both!). This will be satisfied if BDEF -> A and/or C.

III. Transactions/CC [17 points]

T1	R(A)						R(C)	W(B)	COM		
T2				R(A)	W(A)	COM					
T3		R(A)	R(C)							W(B)	COM

1. [4 points] Consider the preceding schedule for three transactions. R indicates a read of a page, W indicates a write of a page, and COM indicates a commit.
    a. Draw the arrows for the dependency graph for this schedule.
    b. Is the schedule conflict serializable?

    a. The dependency graph is shown below.



    b. Yes, it is conflict serializable, because the dependency graph has no directed cycles.

2. [5 points] Which of the following changes to the above schedule will result in a schedule that is possible using strict two-phase locking? (**Mark all that apply**)
    A. Make T1 abort instead of commit
    B. Make T2 abort instead of commit
    C. Remove R(A) from T1 and T3
    D. Remove T2 from the schedule
    E. Make T3 write to a page D instead of page B

    C,D

    A and B are incorrect because strict 2PL deals with acquisition of locks before a transaction commits or aborts. C is correct because if T1 and T3 do not acquire a shared lock to read page A, then T2 can acquire an exclusive lock to write page A before T1 and T3 commit. Similarly, D is correct because if we remove T2 from the schedule, then no transaction needs an exclusive lock on page A. E is incorrect because T2 and T3 can already acquire an exclusive lock to write page B, and even if T3 writes to page D instead of page B, strict 2PL still prevents T2 from completing.

T1	R(A)	R(B)							W(B)
T2			R(A)	R(B)	W(A)				
T3						R(B)	R(C)	W(C)	

3. [3 points] Consider the preceding schedule. A, B, and C are positive integers. Each transaction reads in two integers, adds them, and writes the result. Which of the following statements is/are true? (**Mark all that apply**)
    A. The schedule avoids cascading aborts
    B. The schedule is conflict serializable
    C. The schedule is equivalent to some serial schedule

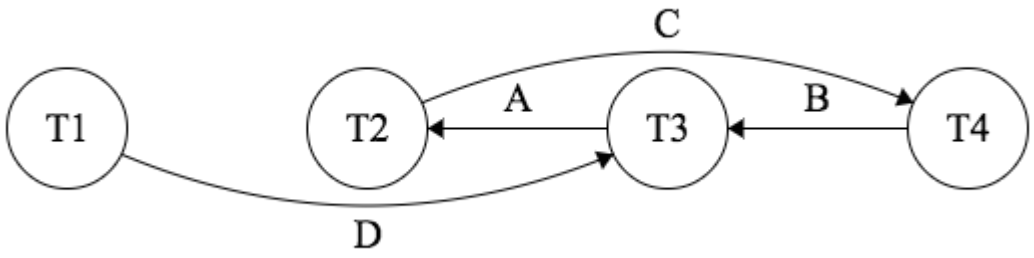
    A

    A is correct because after each page is written, no transaction reads that page, so if any transaction aborts, then no other transactions need to abort. B is incorrect because in the dependency graph, there is an edge from T1 to T2 (A), and an edge from T2 to T1 (B), so the graph has a directed cycle. C is incorrect because there is no way to interleave the reads and writes of the transactions and have the same outputs in integers A, B, and C.

T1	RS(D)							RX(D)	
T2		RS(A)							RS(C)
T3			RS(D)	RX(B)	RX(A)				
T4						RX(C)	RS(B)		

4. [5 points] Assume that we are using strict two-phase locking, and the objects being locked are pages. RS indicates a request for a shared lock on a page, and RX indicates a request for an exclusive lock on a page. No transactions commit or abort during this time.
    a. Draw the arrows for the waits-for graph at the end of this schedule.
    b. Does the schedule lead to deadlock?

    a. The waits-for graph is shown below.



    b. Yes, it leads to deadlock, because there is a cycle in the waits-for graph.

IV. Text Search [14 points]

We have a search engine with a corpus containing **only** the documents
Document ID 1: “A time to plant and a time to reap”
Document ID 2: “Time for you and time for me”
Document ID 3: “Time flies”

1a. [2 points] Given that the stemmed version of the word “flies” is the term “fly”, what is the TF-IDF of “fly” in document 3?

1 log (3)
“fly” appears once (stemmed from “flies”) in document 3, so its TF is 1. It appears only in document 3, so its IDF is log (3/1) = log (3).

1b. [2 points] What is the TF-IDF of “time” in document 1?

2 log (1)
“time” appears twice in document 1, so its TF is 2. It appears in three documents, so its IDF is log (3/3) = log (1).

2. [4 points] We want to perform cosine similarity ranking on these documents, so we represent each document with a vector containing the TF-IDFs of different terms in our documents. Which of the following lists of terms would allow us to correctly compute cosine similarity? Assume that standard stop words include “a”, "and", "to", and "for". (**Mark all that apply**)

A. (“plant”, “you”, “fly”, “me”, “reap”, “time”)
B. (“time”, “plant”, “reap”, “you”, me”, “fly”)
C. (“time”, “plant”, “you”, “me”, “fly”)
D. (“plant”, “reap”, “you”, “me”, “fly”)

A, B, D
All we care about when making cosine similarity vectors are the sets of terms themselves, so permuting the orderings of terms in a vector will not influence cosine similarity. (You may also observe that a dot product is agnostic to the ordering of terms in a vector.) After we remove stop words and perform stemming, lists A and B contain all the relevant terms that we would care about in some query against this corpus.
Not all the terms need to be present, however. In fact, list D is sufficient for computing cosine similarity: with careful observation (question 1(b) is a hint) you could realize that since “time” showed up in every document, its TF-IDF would always be 0, and consequently “time” cannot affect the result of our cosine similarity ranking.
List C was not a valid answer; the absence of “reap” would for example cause queries including that term to rank improperly.

3. [4 points] Suppose we issue our engine the query “plant OR fly”. In the spaces provided, write **only** the document IDs which are returned by this query, in order of relevance **as determined by cosine similarity**. (You may want to leave some spaces **blank**.) Break ties with smaller document IDs first, and then larger document IDs.

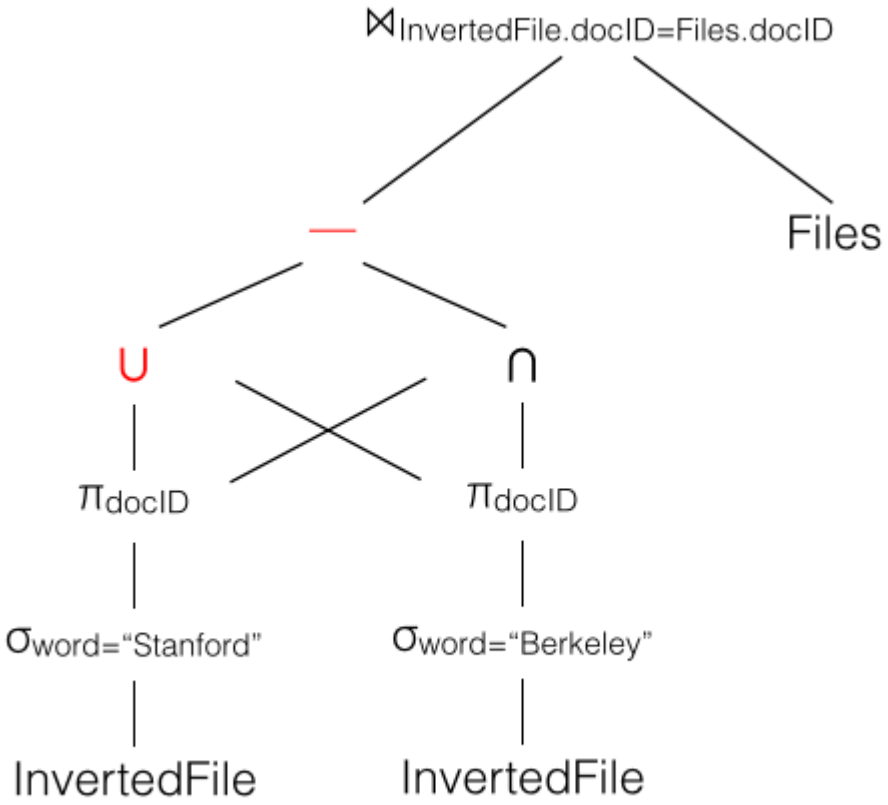
3, 1, <blank>

First, observe that document 2 is never returned by our query: it contains neither the term “plant” nor the word “fly”.
Now we can compute cosine similarity between the document “plant OR fly” and both documents 1 and 3. This would involve computing the TF-IDFs of each of the relevant terms in documents 1 and 3 and then normalizing the resulting vector, after which you would compute a dot product on these vectors with the query document.
Instead of doing the math to find these vectors, you could also have noticed that “plant” and “fly” have the same TF-IDF in documents 1 and 3, but document 1 has another “important” term in it (“reap”) while “fly” is the only important term in document 3 (discounting “time” in either document). Then since “plant” and “fly” had the same TF-IDF, but document 1 was “longer”, the TF-IDF of “plant” would have been scaled by a smaller normalizing factor than “fly”.

4a. [2 point] For this question, assume that we have a text index with the following schema:

```
Files(docID text, content text)
InvertedFile(word text, docID text)
B+-tree Alternative 2 on Files
B+-tree w/"postings list" at leaves on InvertedFile.word
```

If we have two terms A and B, the exclusive OR operator "XOR" returns all documents which contain *either* term "A" or term "B", but not *both*. Suppose we run the query "Berkeley XOR Stanford" on our search engine. Fill in the spots on your answer sheet corresponding to the missing parts of the generated query plan below (indicated by (i) and (ii))



4b. [2 points] One of the following join algorithms would be ideal for evaluating the logical operator at the top of our query plan $\bowtie_{\text{InvertedFile.docID=Files.docID}}$. Which one should we choose? (You can ignore issues of parallelism when answering this question, though they don't really affect the answer.)

A. Block Nested Loops join
B. Index Nested Loops join

C. Hash join
D. Sort Merge join

B

As specified above, and as is usual for text search engines, our Files relation is conveniently indexed (by a B+ tree). Index Nested Loops Join will work best in this scenario, streaming result document IDs against this index.

V. Query Optimization [17 points]

Suppose the "System R" assumptions about uniformity and independence from lecture hold. Assume that costs are estimated as a number of I/Os, without differentiating random and sequential I/O cost.

Consider the following relational schema and statistics:

```
Students    (sid, name, age, gpa, year)
Courses     (cid, name, professor)
Enrollment  (sid, cid, credits)
```

	Tuples	Tuples / Page	Ranges	Distinct Values	Indexes
`Students`	40,000	20	SID: 0 to 40,000 age: 15 to 44	gpa: 500 age: 30	unclustered B+-tree on age (800 pages, d= 4) clustered B+-tree on sid (400 pages, d = 3)
`Enrollment`	60,000	50	-	-	clustered B+-tree on cid (200 pages, d = 2)
`Courses`	800	40	-	professor: 100 name: 200	unclustered B+-tree on prof (100 pages, d = 3)

1. Suppose we run the following query. (<> is the SQL syntax for "not equals").

```
SELECT *
  FROM Courses
 WHERE name <> "CS 186"
   AND professor = "Hellerstein";
```

a. [2 points] What is the selectivity of each conjunct in the WHERE clause?

name: .995 (199/200)
professor: .01 (1/100)

b. [2 points] Which single table access plan would we choose?

A) Sequential/File Scan on Courses
B) Sequential/File Scan on Enrollment
C) Index scan of unclustered B+Tree on Courses.professor
D) Index scan of clustered B+Tree on Enrollment.cid

2. [6 points] Suppose we run the following query. On the line labeled “considered”, mark the letters of *all* 2-table subplans that will be considered. On the line labeled “chosen”, mark the letters of *all* 2-table subplans that are chosen. Ignore interesting orders.

```
SELECT *
  FROM Students S, Courses C, Enrollment E
 WHERE S.age > 40
   AND C.name LIKE "CS%"
   AND S.sid = E.sid
   AND E.cid = C.cid;
```

A) S ⋈ C (200,000 IOs)

B) C ⋈ S (300,000 IOs)

C) S ⋈ E (400,000,000 IOs)

D) E ⋈ S (500,000,000 IOs)

E) C ⋈ E (300,000 IOs)

F) E ⋈ C (400,000 IOs)

Considered: C, D, E, F
Chosen: C, E

3. [3 points] Now, mark the letters for *all* three-table access plans that would be considered, and write down the letter of the final query plan that would be chosen. (Cost given are cumulative for the full query. Again, ignore interesting orders in formulating your answer.)

A) E ⋈ (S ⋈ C) (3,900,000,000 IOs)

B) E ⋈ (C ⋈ S) (6,000,000,000 IOs)

C) (C ⋈ S) ⋈ E (5,000,000,000 IOs)

D) (S ⋈ C) ⋈ E (8,000,000,000 IOs)

E) C ⋈ (S ⋈ E) (7,000,000,000 IOs)

G) (S ⋈ E) ⋈ C (7,500,000,000 IOs)

H) (E ⋈ S) ⋈ C (8,000,000,000 IOs)

I) S ⋈ (C ⋈ E) (3,500,000,000 IOs)

J) S ⋈ (E ⋈ C) (6,000,000,000 IOs)

K) (C ⋈ E) ⋈ S (4,000,000,000 IOs)

F) C ⋈ (E ⋈ S) (9,000,000,000 IOs) L) (E ⋈ C) ⋈ S (10,000,000,000 IOs)

Considered: G, K
Chosen: K

4. [4 points] **True or False:**

a. If we had considered interesting orders in the query of Question 3, it would have further pruned the number of plans chosen. False
b. In Question 3, it would be beneficial to consider interesting orders on C.name during optimization. False
c. In Question 3, it would be beneficial to consider interesting orders on C.cid during optimization. True
d. A modern optimizer should consider “interesting hashes”, since the output of a hash-based operator (e.g. hash join) is organized in a way that would decrease the cost of performing a subsequent hash-based operator (e.g. group-by). True

CS411 Database Systems

Fall 2005, Prof. Chang

Department of Computer Science
University of Illinois at Urbana-Champaign

Final Examination

December 14, 2005
Time Limit: 180 minutes

- Print your name and NetID below. In addition, print your NetID in the upper right corner of every page.

  Name: _____ **NetID:** _____

- Including this cover page, this exam booklet contains **11** pages. Check if you have missing pages.
- The exam is closed book and closed notes. You are allowed to use scratch papers. No calculators or other electronic devices are permitted. Any form of cheating on the examination will result in a zero grade.
- Please write your solutions in the spaces provided on the exam. You may use the blank areas and backs of the exam pages for scratch work.
- Please make your answers clear and succinct; you will lose credit for verbose, convoluted, or confusing answers. *Simplicity does count!*
- Each problem has different weight, as listed below– So, plan your time accordingly. *You should look through the entire exam before getting started, to plan your strategy.*
- Problems that are related to homework (*e.g.*, in terms of concepts covered) are marked with [*HW*].

Problem	1	2	3	4	5	6	7	8			Total
Points	12	18	10	10	10	10	15	15			100
Score											
Grader											

Problem 1 (*12 points*) Misc. Concepts

For each of the following statements, indicate whether it is *TRUE* or *FALSE* by circling your choice. You will get *1 point* for each correct answer, *-0.5 point* for each incorrect answer, and *0 point* for each answer left blank.

(1) *True* *False*

Transaction management was one of the concepts Ted Codd created in his seminal work of defining the relational model.

(2) *True* *False*

A relation R in 3NF is also in 4NF.

(3) *True* *False*

The second and higher levels must be sparse in an index of multiple levels on sequential files.

(4) *True* *False*

Hash index is efficient in answering range queries, such as "finding products with price higher than 100".

(5) *True* *False*

The order of insertions into a B+ tree will affect the tree's final structure at the end.

(6) *True* *False*

The time needed to access a page on disk is comprised of two components: disk seek time and data transfer time.

(7) *True* *False*

If possible, we should always use a bushy join tree instead of a left-deep join tree, because the former is more efficient than the latter.

(8) *True* *False*

In SQL, without GROUP BY, we cannot use HAVING.

(9) *True* *False*

We can optimize query plans by pushing a selection down an expression tree, but not by moving a selection up the tree.

(10) *True* *False*

Query optimization is a major concept that enables declarative SQL queries, because it automatically generates query plans, without having users to write procedural queries.

(11) *True* *False*

For buffer management, a policy like LRU may be *inefficient* for transaction processing in RDBMS, but it will not affect *correctness*.

(12) *True* *False*

Pointer swizzling refers to the techniques that convert disk pointers to memory addresses.

Problem 2 (*18 points*) Short Answer Questions

For each of the following questions, write your answer in the given space. You will get *2 points* for each correct answer.

(1) Answer: _____
Write the following query in relational algebra, for relations $R(a,\ b)$ and $S(a,\ b)$:
(SELECT a FROM R WHERE b<10) EXCEPT (SELECT a FROM S WHERE b>5).

(2) Answer: _____
Write the following relation algebra expression in SQL, for relations $R(a,\ b)$ and $S(a,\ c)$: $\pi_{b,c}((\sigma_{b>10}R)\bowtie(\sigma_{c>5}S))$

(3) Answer: _____
Suppose one block can hold 10 pointers or 20 data tuples. Given a relation of 10000 tuples, how many blocks do we need for a 2-level index of this relation (given that the first level is sparse).

(4) Answer: _____
Consider relation R with five attributes ABCDE, and the following functional dependencies: $A \rightarrow B, BC \rightarrow D, D \rightarrow E$. Give the complete closure for {AC}.

(5) Answer: _____
A professor record consists of one fixed length field **SSN** and one variable length field **name**. The records are augmented by an additional repeating field that represents the courses a professor teaches. Each course is of fixed length. Show the layout of a professor record if the variable length field and repeating courses are kept within the record itself.

(6) Answer: _____
Transactions should be "ACID". What does "A" stand for?

(7) Answer: _____
In one sentence, explain the *principle of optimality*, which governs the correctness of dynamic programming.

(8) Answer: _____
Consider two relations R(a,b,c) and S(a,d,e). T(R)=200, T(S)=100. R.a is a foreign key referencing S.a and S.a is the primary key of S. What is the estimated size of R⋈S?

(9) Answer: _____
Among the four properties "ACID" achieved by transaction management, which properties are guaranteed by failure recovery?

Problem 3 (*10 points*) Schema Decomposition [*HW*]

Consider a relation R with five attributes ABCDE. The following dependencies are given: $A \rightarrow B$, $BC \rightarrow E$, $ED \rightarrow A$.

(a) List all keys for R. (3 points)

(b) Is R in 3NF? Briefly explain why. (3 points)

(c) Is R in BCNF? If yes, please explain why. Otherwise, decompose R into relations that are in BCNF. (4 points)

Problem 4 (*10 points*) Query Languages [*HW*]

The following questions refer to the database schema below: *Product*(*pid, price*), *Order*(*oid, cid, pid, quantity*), *Customer*(*cid, name, age*).

(a) Write a query, in *relational algebra*, to return the names of senior customers (older than 60). (2 points)

(b) Write a query, in *relational algebra*, to return the names of customers who didn't order any product. (3 points)

(c) Write a SQL query, to list each product along with its total ordered quantities, in descending order of the amount. (5 points)

Problem 5 (*10 points*) Indexing: B+tree [*HW*]

Consider constructing a B+-tree of order 3 (*i.e.*, $n = 3$, each index node can hold n keys and $n+1$ pointers).

(a) Show the resulting tree after inserting keys in this order: $40, 10, 50, 30, 90, 80, 70, 20, 60, 100$. (4 points)

(b) Based on the tree in (a), show the steps in executing the following operation: Lookup all records in the range 20 to 60. (3 points)

(c) Show the resulting tree after deleting key 60 from the tree in (a). (3 points)

Problem 6 (*10 points*) Query Processing [*HW*]

Consider joining two relations $R(x,y)$ and $S(x,z)$ on their common attribute x. The size of relation R is 150 blocks and the size of relation S is 100 blocks. Attribute x has 50 different values and is evenly distributed in both R and S. Suppose that both relations are not sorted by attribute x.

(a) Suppose the memory buffer has 15 blocks, compute the cost of join using a block-nested loop join. (3 points)

(b) Suppose the memory buffer has 15 blocks, compute the cost of join using a sort-merge join. (3 points)

(c) Can you estimate the size of the output relation by joining R and S on x. If yes, estimate it. If no, explain why. (4 points)

Problem 7 (*15 points*) Query Optimization

Consider the following query that joins *Student*(*sid, sname, sdept*), *Enrollment*(*sid, cid*), *Courses*(*cid, ctitle, iid*), *Instructor*(*iid, iname, iaddr*).

select *sname*, *ctitle*, *iname*
from *Student S*, *Enrollment E*, *Course C*, *Instructor I*
where join-conditions AND selection-conditions

In the **where**-clause, we have the following conditions:

- The join-conditions specify how the relations are joined. In this query, they are fixed to natural joins, *i.e.*: $S.sid = E.sid$ AND $E.cid = C.cid$ AND $C.iid = I.iid$.
- The selection-conditions are of the form c_1 AND c_2 AND $\cdots$ AND c_n, where each c_i is a selection condition on some relation, *e.g.*, $S.sdept =$ "CS" or $I.iaddr$="1234SC".

(a) Use this example, explain why join ordering is important for minimizing the cost of a query plan. Give an example scenario to support your explanation. (*5 points*)

(b) Consider dynamic programming for generating the optimal join order. Without actually working through the whole process, calculate how many subqueries each iteration needs to consider. (Note: we are asking for logical queries, **NOT** physical plans) (*5 points*)

(c) Now, if in the dynamic programming process, you do not want to consider cartesian products at all. With this assumption, give your answers for (b) again here. (*5 points*)

Problem 8 (*15 points*) Failure Recovery [*HW*]

Consider the following logs. Note that, in this log sequence, we also show various Output actions (at the time points when the actions are carried out).

Log ID	Log
1	$\langle$START $T1\rangle$
2	$\langle T1, A, 10\rangle$
3	$\langle$START $T2\rangle$
4	$\langle T1, B, 10\rangle$
5	$\langle$COMMIT $T1\rangle$
6	$\langle T2, B, 10\rangle$
7	$\langle$COMMIT $T2\rangle$
	// Output(A); Output(B);
8	$\langle$START $T3\rangle$
9	$\langle T3, A, 10\rangle$
10	$\langle$START $T4\rangle$
11	$\langle T3, B, 20\rangle$
12	$\langle$COMMIT $T3\rangle$
13	$\langle T4, C, 10\rangle$
14	$\langle$START $T5\rangle$
	// Output(A); Output(B);
15	$\langle$COMMIT $T4\rangle$
	// Output(C)
16	$\langle T5, D, 10\rangle$
17	$\langle$COMMIT $T5\rangle$
	// Output(D)

(a) Can this be an *undo* log? Explain why or why not. (*3 points*)

Note: For the questions below, let's assume that this log sequence is a *redo* log.

(b) Suppose we want to perform the outputs *as early as possible*, show on the figure how we should change the output actions. (*4 points*)

(c) Suppose we want to start checkpointing right after Action 13: First, show on the figure this start checkpointing log record. (*2 points*) Then, show on the figure the end checkpointing log record. (*2 points*)

(d) Continue from above. Suppose the system crashes after Action 16. Specify the log ID that we need to start to do *redo* actions. (*4 points*)

CS411 Database Systems

Fall 2005, Prof. Chang

Department of Computer Science
University of Illinois at Urbana-Champaign

Final Examination

December 14, 2005
Time Limit: 180 minutes

Problem 1 (*12 points*) Misc. Concepts

(1) False

(2) False

(3) True

(4) False

(5) True

(6) False

(7) False

(8) True

(9) False

(10) True

(11) False

(12) True

Problem 2 (*18 points*) Short Answer Questions

(1) $\pi_a(\sigma_{b<10}R) - \pi_a(\sigma_{b>5}S)$

(2) SELECT b, c FROM R, S WHERE $R.a = S.a$ AND $R.b$>10 AND $S.c$>5

(3) 55

(4) ABCDE

(5) Refer to Figure 1

(6) Atomic (or Atomicity)

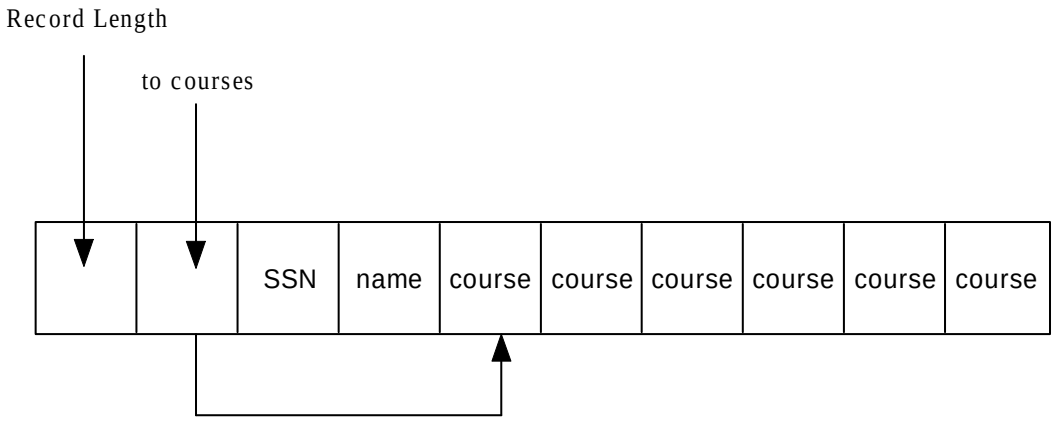


Figure 1: A Professor Record

(7) Optimal plan must come from optimal subplans.

(8) 200

(9) Atomic and Durable (AD)

Problem 3 (*8 points*) Schema Decomposition

(a) CDE, ACD, BCD

(b) Yes, it is in 3NF because B, E and A are all parts of keys.

(C) No. For example, $BC \rightarrow E$ violates BCNF. If we decompose using this FD, we get BCE, ABCD. Further decomposing ABCD using $A \rightarrow B$ we get AB, ACD. Therefore, it could be decomposed into BCE, AB, ACD.

Problem 4 (*9 points*) Query Languages

(a) $\pi_{name}(\sigma_{age>60}(Customer))$

(b) $\pi_{name}(Customer) - \pi_{name}(Order \bowtie Customer)$

(c) SELECT Product.pid, SUM(quantity) FROM Product LEFT OUTER JOIN Order GROUP BY Product.pid ORDER BY SUM(quantity) DESC

Problem 5 (*10 points*) Indexing: B+ tree

(a) Refer to Figure 2.

(b) $30 < 40 \rightarrow$ 1st pointer, goes to 1st leaf, get record with key 20, follow the chain to the 3rd leaf, getting records with keys 30, 40, 50, 60.

(c) Either Figure 3 or 4 is ok.

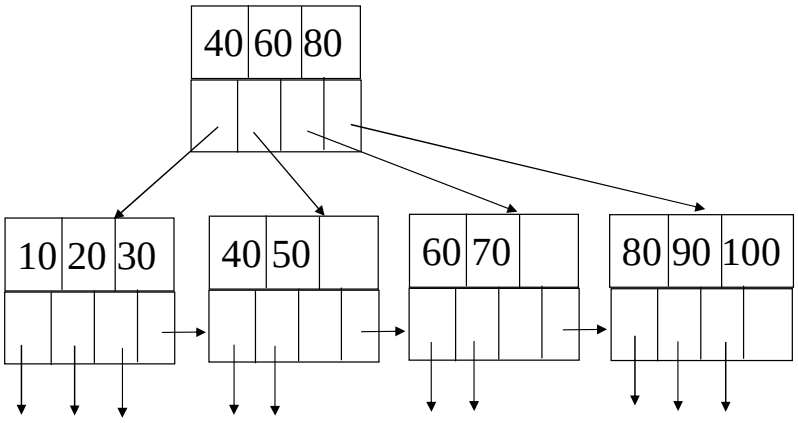


Figure 2: B+ tree after insertion

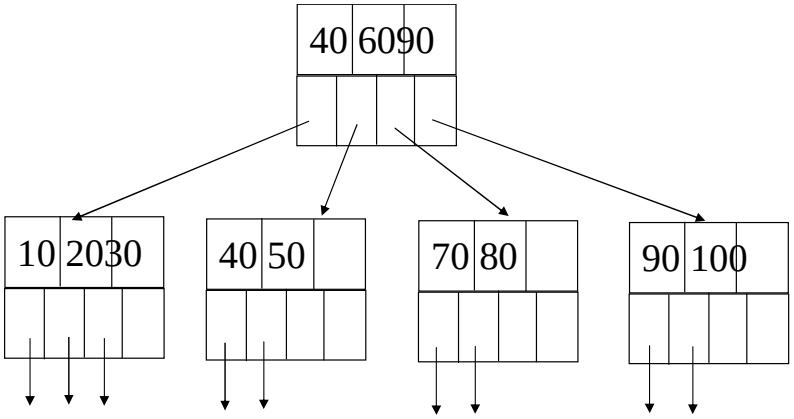


Figure 3: B+ tree after deletion

Problem 6 (*10 points*) Query Processing

(a) $150 + \lceil 150/(14-1) \rceil * 100 = 1250$ blocks

or

$100 + \lceil 100/(15-1) \rceil * 150 = 1300$ blocks

(b) 5(B(R)+B(S)) = 1250 blocks

(c) No. Here we only know the number of blocks for R and S. We need the number of tuples for R and S to estimate the size of the output relation.

Problem 7 (*19 points*) Query Optimization

(a) Joining in different orders will result in different sizes of intermediate relations, thus different processing costs. As an example, suppose the sizes of Student, Enrollment, and Courses are 100, 1000, 1000, respectively. If we join Student with Courses first, which is a cartesian

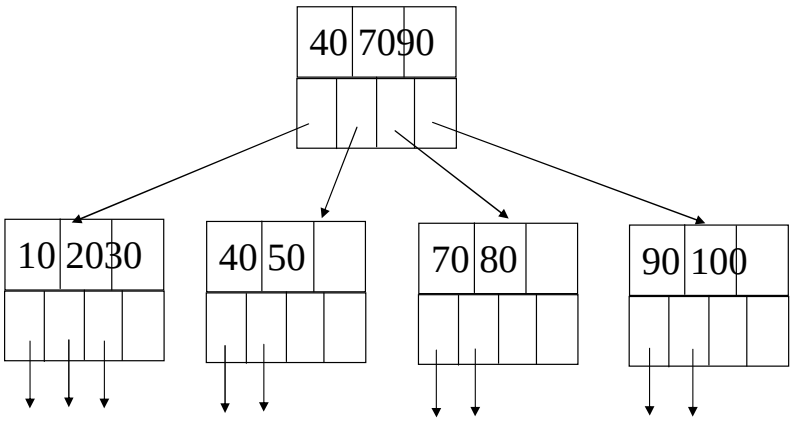


Figure 4: B+ tree after deletion (Alternative)

product, the size of the intermediate results is 100000. We then join it with Enrollment. However, we can also join Student with Enrollment first, with the condition S.sid=E.sid, resulting in 1000 intermediate tuples, which is much smaller than 100000. We then join the intermediate results with courses.

(b) Iteration 1, queries with 1 table: Student, Enrollment, Courses, Instructor. Totally 4.

Iteration 2, queries with 2 tables: SE, SC, SI, EC, EI, CI. Totally 6.

Iteration 3, queries with 3 tables: SEC, SEI, SCI, ECI. Totally 4.

Iteration 4, queries with 4 tables: SECI. Totally 1.

(c) Iteration 1, queries with 1 table: Student, Enrollment, Courses, Instructor. Totally 4.

Iteration 2, queries with 2 tables: SE, EC, CI. Totally 3.

Iteration 3, queries with 3 tables: SEC, ECI. Totally 2.

Iteration 4, queries with 4 tables: SECI. Totally 1.

Problem 8 (*14 points*) Failure Recovery

Log ID	Log
1	$\langle$START $T1\rangle$
2	$\langle T1, A, 10\rangle$
3	$\langle$START $T2\rangle$
4	$\langle T1, B, 10\rangle$
5	$\langle$COMMIT $T1\rangle$
—————	// Output(A); Output(B);
6	$\langle T2, B, 10\rangle$
7	$\langle$COMMIT $T2\rangle$
—————	// Output(B);
	// Output(A); Output(B);(move up)
8	$\langle$START $T3\rangle$
9	$\langle T3, A, 10\rangle$
10	$\langle$START $T4\rangle$
11	$\langle T3, B, 20\rangle$
12	$\langle$COMMIT $T3\rangle$
—————	// Output(A); Output(B);
13	$\langle T4, C, 10\rangle$
—————	// <START CKPT(T4)>;
14	$\langle$START $T5\rangle$
	// Output(A); Output(B);(move up)
—————	// <END CKPT>;
15	$\langle$COMMIT $T4\rangle$
	// Output(C)
16	$\langle T5, D, 10\rangle$
17	$\langle$COMMIT $T5\rangle$
	// Output(D)

(a) No. It can't be undo log. Undo log requires data values be written to disk before the corresponding transactions commit log record is written to disk. There are many counterexamples here.

(b) Output values immediately after the commit log record is written to disk.

(c) Please refer to the log shown above. The <END CKPT> could be put anywhere after log 15 as well. But it is better to put it as early as possible.

(d) We need to redo T4, starting from log 13.

CS411 Database Systems

Fall 2007

Department of Computer Science
University of Illinois at Urbana-Champaign

Final Examination

December 12, 2007
Time Limit: 180 minutes

- Print your name and NetID below. In addition, print your NetID in the upper right corner of every page.

Name: _____ **NetID:** _____

- Including this cover page, this exam booklet contains **17** pages. Check if you have missing pages.
- The exam is closed book and closed notes. You are allowed to use scratch papers. No calculators or other electronic devices are permitted. Any form of cheating on the examination will result in a zero grade.
- Please write your solutions in the spaces provided on the exam. You may use the blank areas and backs of the exam pages for scratch work.
- Please make your answers clear and succinct; you will lose credit for verbose, convoluted, or confusing answers. *Simplicity does count!*
- Each problem has different weight, as listed below– So, plan your time accordingly. *You should look through the entire exam before getting started, to plan your strategy.*

Problem	1	2	3	4	5	6	7	8			Total
Points	15	10	10	14	17	12	12	10			100
Score											
Grader											

Problem 1 (*15 points*) Basics

For each of the following statements, indicate whether it is TRUE or FALSE by circling your choice. If you change your mind, cross out both responses and write "True" or "False." You will get 1 point for each correct answer, 0 point for each incorrect answer.

(1) *True False*
Logical and physical addresses are both representations for the database address.

(2) *True False*
When a client requests a record that contains a BLOB (binary, large object), the database server that receives the request should return the entire record at a time.

(3) *True False*
When an index covers many blocks, we may want to put a second-level index on the first-level index that contains pointers to records. When we use multiple levels of indexes, the first-level index must be sparse.

(4) *True False*
Secondary indexes are always dense.

(5) *True False*
In B-trees, we sometimes need to have overflow blocks.

(6) *True False*
Dynamic hash tables support range queries.

(7) *True False*
In linear hash tables, we sometimes have overflow blocks.

(8) *True False*
One-pass algorithms for set union of two relations R and S requires M main memory buffers such that $B(R) + B(S) \leq M$.

(9) *True False*
In algebraic laws, $\sigma_C(R - S) = R - \sigma_C(S)$ holds.

(10) *True False*
Dynamic programming is a bottom-up method, where we consider only the best plan for each subexpression of the logical-query plan.

(11) *True False*
In undo logging, it is not always possible to recover some consistent state of a database system if the system crashes during recovery.

(12) *True False*
A scheduler based on the two-phase locking scheme always executes multiple transactions with some serial schedule.

(13) *True False*
Two-phase locking prevents deadlocks.

(14) *True False*
Concurrency control by timestamps is superior to that by locks if most transactions are read-only.

(15) *True False*
ACR (avoids cascading rollback) schedules are always serializable.

Problem 2 (*10 points*) Pointer Swizzling

Suppose that the important actions related to data storage take the following times, in some arbitrary time units:

- On-demand swizzling of a pointer: 30;
- Automatic swizzling of pointers: 20 per pointer;
- Following a swizzled pointer: 1;
- Following an unswizzled pointer: 10.

(i) Suppose we design a pointer-swizzling control scheme like the following. At the beginning, we automatically swizzle 30% of the pointers and leave the rest unswizzled. Once a pointer is followed, we swizzle it by probability 0.5. If an unswizzled pointer has been followed twice, we swizzle it. Suppose there are 200 pointers in our data. The number of times that they are followed by a program is distributed according to the following histogram.

times of being followed	0	1	2	3
number of pointers	20	80	60	40

What's the **expected cost** of this program in terms of pointer following? (6 points)

(ii) A block in memory is said to be *pinned* if it cannot at the moment be written back to disk safely. Explain in what situation a block could be pinned because of swizzled pointers and describe how we should extend a translation table that maps database addresses to memory addresses in order to unpin such blocks. (4 points)

Problem 3 (*10 points*) Indexing

Consider the B-tree of $d = 2$ (*i.e.*, each index node can hold at least $d = 2$ keys and at most $2d = 4$ keys) shown in Figure 1.

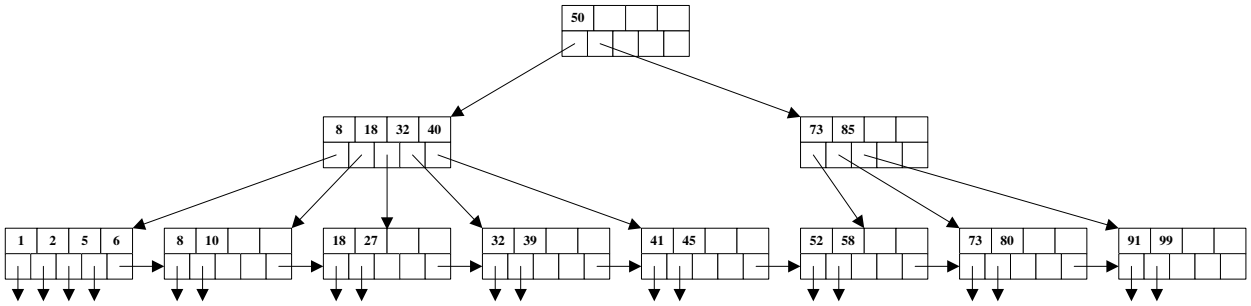


Figure 1: B-tree of Problem 3

(a) What is the largest number of records that can be inserted into this tree (more nodes may be created), while the height of the tree does not change. (3 points)

(b) show the steps in executing the following operation: Lookup all records in the range from 10 to 58 (including 10 and 58). (3 points)

(c) Show the B+ tree that would result from deleting the data entry with key 91 from the original tree. (4 points)

Problem 4 (*14 points*) Query Execution

Consider two relations $R(a, b)$ and $S(b, c)$ with the following statistics:

$T(R) = 10,000$, $B(R) = 1,000$ (each block contains 10 tuples),
$V(R, b) = 200$ (number of distinct values of attribute b in R),

$T(S) = 6,000$, $B(S) = 1,500$ (each block contains 4 tuples),
$V(S, b) = 100$ (number of distinct values of attribute b in S),
$V(S, c) = 20$ (number of distinct values of attribute c in S) and $\underline{c > 100}$.

Also, we assume the number of available memory blocks is $M = 101$.

Please answer the following questions:

(i) Estimate the number of tuples in $\sigma_{c=150}(S)$ (2 points)

(ii) Estimate the number of tuples in $R \bowtie \sigma_{c>25}(S)$. (2 points)

(iii) Suppose we have a B-tree index available for attribute b of S. The tree has 3 levels, and each node contains 4 keys– that is, there would be totally 100 keys in the 5×5 leaf nodes, where each key would correspond to a distinct value of b in S. Assume each node of the index occupies one block.

For simplicity, we assume the tuples with the same b value are stored consecutively in the disk but may spread in different blocks. Please estimate the worst cost of executing the following query: (the cost is measured by disk I/Os for accessing the index and the tuples). (3 points)

SELECT *
FROM S
WHERE $b = 100$ OR $b = 1000$

(iv) Consider joining R and S using a block-based nested loop join (without using any index). Suppose R is used as the outer loop. Estimate the cost. (3 points)

(v) Now suppose we want to use sort-join. Assuming that the number of tuples with the same b value is not large, we aim to use the more efficient sort-based join approach:

1. Created sorted sublists of size M, using b as the sort key, for both R and S.
2. Bring the first block of each sublist into the memory buffer.
3. Repeatedly find the least b-value, and output the join of all tuples from R with all tuples from S that share this common b-value. If the buffer for one of the sublist is exhausted, then replenish it from disk.

Please estimate the cost (total disk I/Os) of applying this sort-join on R and S. Also, state the requirement on the number of memory blocks M considering the maximum number of blocks K for holding tuples of R and S with the same b value.

(4 points)

Problem 5 (*17 points*) Query Optimization

Suppose we want to compute the following:

$$\tau_b(R(a,b) \bowtie S(b,c) \bowtie T(c,d)), \quad \text{where } \tau_b \text{ specifies sorting on } b.$$

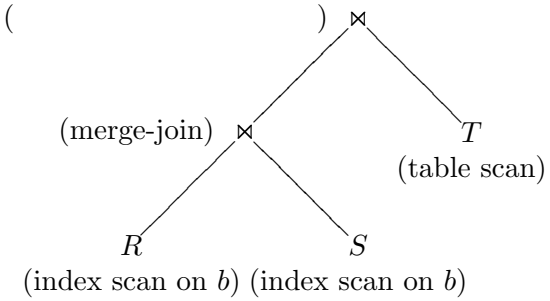
That is, we want to join three relations R, S, and T, and sort the results on attribute b. Let us make the following assumptions:

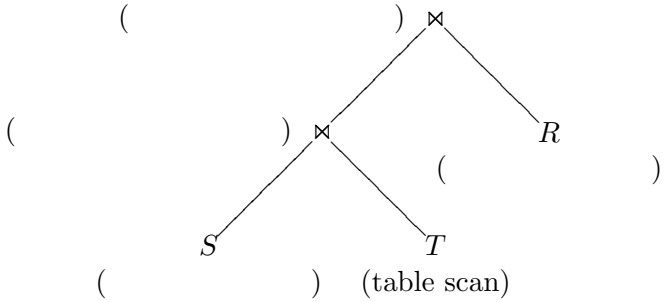
- First, for accessing each relation:
  - R can be *index scanned* on attribute a or b, or *table scanned*.
  - S can be *index scanned* only on b, or *table scanned*.
  - T can only be *table scanned*.

  We assume the index is based on B-tree. Accessing a relation using *index scan* on an attribute will produce records sorted by that attribute, while *table scan* does not give any guaranteed output order for the records.
- Second, for joining two relations, we have the following two choices:
  - *merge-join* can be used if the two relations are already sorted on the join attribute. It will simply join the two sorted relations using the merge part of the simple sort-based join algorithm, and the output would be sorted.
  - *nested-loop join* can be used in any case. For simplicity, we assume the block-based nested-loop join is used and the relation produced in the *left* sub-tree of a query plan will be used as the outer loop.

(i) Please draw *two* logical query plans (without concerning the physical access and join methods) that are *not* "left-deep." (3 points)

(ii) Now, consider the following two logical query plans (regardless of what your answer is for (i)), where we also fill in some physical access and join methods. Note we show only the join part (and not sorting) of the query.





Suppose we would like to avoid applying an additional sort on the final output relation, while still achieve the desired sorted result (i.e., τ_b). Please help fill appropriate access and join methods into the brackets in the above two query plans. (5 points)

(iii) An *interesting order* holds for a join result if it is sorted in an order that will be useful for the operations in the higher part of the expression tree– for example, sorted on the attribute(s) specified in a sort (τ) operator at the root, or the join attribute(s) of a later join.

For our problem, we want the final result sorted on attribute b. Consider the following three plans for $R \bowtie S$. Please circle "yes" if the plan produces an interesting order, and "no" if not. (3 points)

Plan	$R \bowtie S$	interesting order?
Plan A	(table-scan R) nested-loop-join (table-scan S)	yes / no
Plan B	(index-scan R on b) merge-join (index-scan S on b)	yes / no
Plan C	(index-scan R on b) nested-loop-join (table-scan S)	yes / no

Please briefly explain your choices. (3 points)

(iv) For query optimization, we use an enhanced method that improves upon the dynamic programming approach: It will keep for each subexpression the plan of lowest cost (as dynamic programming does). In addition, it wll also keep the one with lowest cost from those plans that produce an interesting order.

The table below lists the estimated costs for the three plans (see the table in (iii)). Which plans would be kept when considering the subexpression $R \bowtie S$ using this enhanced method? Please circle "yes" if the plan will be kept, and "no" if it will be pruned. (You may need to refer to the results, the interesting order of each plan, in (iii) to make the decisions)

Plan	estimated cost	keep?
Plan A	2000	yes / no
Plan B	3000	yes / no
Plan C	4000	yes / no

Please briefly explain your choices. (3 points)

Problem 6 (*12 points*) Recovery

Consider a database with data items {A, B, C, D}. The system uses an undo/redo scheme and has the following logs. Note that an entry <T, X, old, new> means transaction T changes the value of X from old to new. We consider recovery using this undo/redo log.

1. <T2 start>
2. <T2, B, 10, 11>
3. <T1 start>
4. <T2 commit>
5. <T1, A, 20, 21>
6. <Checkpoint start; Active= {T1}>
7. <T3 start>
8. <T3, C, 30, 31>
9. <T4 start>
10. <T4, B, 40, 41>
11. <T4 commit>
12. <Checkpoint end>
13. <T3, D, 50, 51>
14. <Checkpoint start; Active= {T1, T3}>
15. <T1, C, 31, 32>
16. <T5 start>
17. <T5, D, 51, 52>
18. <T3 commit>
19. <T6 start>
20. <T6, C, 32, 33>
21. <T5 commit>
22. System failed

(i) List all possible values of A, B, C and D. That is, what are the possible data values on the disk at the point of failure (after action 21)? (3 points)

(ii) During recovery, what are the transactions that need to be undone? (3 points)

(iii) During recovery, what are the transactions that need to be redone? (3 points)

(iv) What are the values of A, B, C, D after recovery? Explain why? (3 points)

Problem 7 (*12 points*) Concurrency control

Answer the following questions:

(i) Briefly explain one advantage of nonquiescent checkpointing compared to quiescent checkpointing. (3 points)

(ii) Consider the schedule S: w1(X); w3(X); w2(X); w4(X); r4(Y); w1(Y). Draw the precedence graph for the schedule S. Is it conflict-serializable? Explain why. (3 points)

(iii) Consider the schedule S: r1(X); r2(Y); w2(X); w1(Y). Does deadlock occur using the two-phase locking rule? Explain why. (3 points)

(iv) In the following sequences of events, $R_i(X)$ means that transaction T_i starts and its read set is the list of database elements X. Also, V_i means T_i attempts to validate and $W_i(X)$ means that T_i finishes, and its write set was X. Tell what happens when the following sequence of events is processed by a validation-based scheduler. (3 points)

$$R_1(A,B); R_2(B,C); V_1; R_3(C,D); V_3; W_1(A); V_2; W_2(A); W_3(D)$$

Problem 8 (*10 points*) Deadlocks

Answer the following questions:

(i) For the sequence of actions below, assume that locks are requested immediately before each read and write action. However, if a transaction is already holding a lock on a database element, it does not need to obtain that lock again to execute another action on that element. Also, unlocks occur immediately after the final action that a transaction executes. Tell which locking actions are denied, and whether a deadlock occurs, by drawing a wait-for graph. (5 points)

$$r_1(A); r_2(B); w_1(C); w_2(D); r_3(C); w_1(B); w_4(D); w_2(A);$$

(ii) We observed in our study of lock-based schedules that there are several reasons why transactions that obtain locks could deadlock. Can a timestamp-based scheduler using the commit bit C(X) have a deadlock? If you beleive that the scheduler does not have any deadlock, prove that. Otherwise, describe a situation where a deadlock occurs. (Hint: consider the case where two transactions T_1 and T_2 run while accessing two database elements A and B.) (5 points)

CS411 Database Systems
Fall 2007

Final Exam Solutions

Problem 1 (*15 points*)

(1) True; (2) False; (3) False; (4) True; (5) False;
(6) False; (7) True; (8) False; (9) False; (10) True;
(11) False; (12) False; (13) False; (14) True; (15) False;

Problem 2 (*10 points*)

The distribution of pointers are like the following.

times of being followed	0	1	2	3
Total	20	80	60	40
Automatically swizzled	6	24	18	12
On-demand swizzled after 1st access	N/A	28	21	14
On-demand swizzled after 2nd access	N/A	N/A	21	14

- For the pointers that are never followed: subtotal is 120
  - cost of automatic swizzling of 30%: $6 \times 20 = 120$
- For the pointers that are followed only once: subtotal is 1904
  - cost of automatic swizzling of 30%: $24 \times 20 = 480$
  - cost of following the swizzled pointers: $24 \times 1 = 24$
  - cost of following the unswizzled pointers: $(80 - 24) \times 10 = 560$
  - cost of on-demand swizzling: $28 \times 30 = 840$
- For the pointers that are followed twice: subtotal is 2307
  - cost of automatic swizzling of 30%: $18 \times 20 = 360$
  - cost of following the swizzled pointers: $18 \times 2 + 21 \times 1 = 57$
  - cost of following the unswizzled pointers: $(21 + 21 \times 2) \times 10 = 630$
  - cost of on-demand swizzling: $(21 + 21) \times 30 = 1260$
- For the pointers that are followed three times: subtotal is 1578
  - cost of automatic swizzling of 30%: $12 \times 20 = 240$
  - cost of following the swizzled pointers: $12 \times 3 + 14 \times 2 + 14 \times 1 = 78$
  - cost of following the unswizzled pointers: $(14 + 14 \times 2) \times 10 = 420$
  - cost of on-demand swizzling: $(14 + 14) \times 30 = 840$

The total expected cost is $120 + 1904 + 2307 + 1578 = 5909$.

(ii) Solution: In such situation a block could be pinned because of swizzled pointers: Suppose a block B_1 has within it a swizzled pointer to some data item in block B_2, and we move block B_2 back to disk. Now, should we follow the pointer in B_1, it will lead us to the buffer, which no longer holds B_2. In effect, the pointer has become dangling.

To unpin a block that is pinned because of swizzled pointers from outside, we must "unswizzle" any pointers to it. Consequently, the translation table must record, for each database address whose data item is in memory, the places in memory where swizzled pointers to that item exist. Two possible approaches are:

1. Keep the list of references to a memory address as a linked list attached to the entry for that address in the translation table.

2. If memory addresses are significantly shorter than database addresses, we can create the linked list in the space used for the pointers themselves. That is, each space used for a database pointer is replaced by (a) The swizzled pointer, and (b) Another pointer that forms part of a linked list of all occurrences of this pointer.

Problem 3 (*10 points*)

(a) With height unchanged, this tree can hold at most $5 \times 5 \times 4 = 100$ records. The number of current records is 18. Therefore, this tree can accommodate 82 more records.

(b) $10 < 50 \rightarrow$ 1st pointer, $8 < 10 < 18 \rightarrow$ 2nd pointer, find 10 in the 2nd key. get records with keys 10, 18,27,32,39,41,45,52,58 by following chains.

(c) See Figure 1.

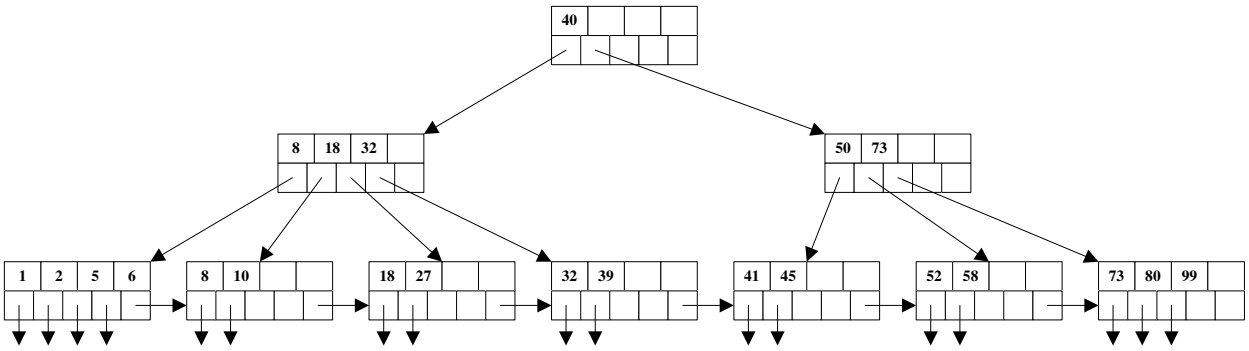


Figure 1: Solution for Problem 4(c)

Problem 4 (*14 points*)

(i) $6,000/20 = 300$

(ii) $T(R)T(S)/\max(V(R,b),V(S,b)) = 10,000 \times 6,000/200 = 300,000$

(iii) Accessing index to find $b = 100$ needs 3 blocks
Accessing index to find $b = 1000$ needs 3 blocks

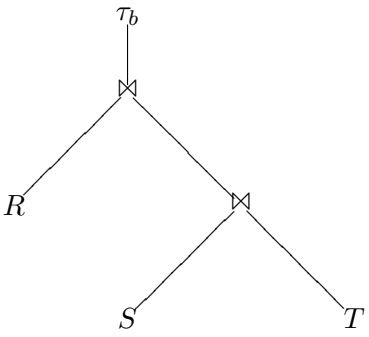
The tuples with $b = 100$ are estimated as $T(S)/V(R, b) = 60$. These will occupy at best 15 blocks but at worst 16 blocks.
Similarly, tuples for $b = 1000$ will occupy at worst 16 blocks.
So, the worst cost of this query should be 3 + 3 + 16 + 16=38 blocks.

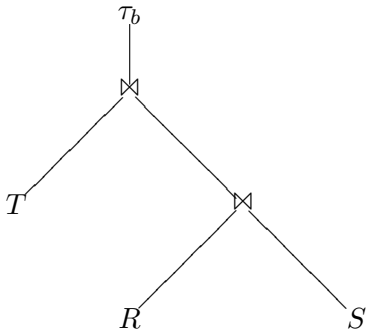
(iv) We shall use 100 blocks to buffer R. Thus, for each iteration, we do 100 disk I/O's to read the chunk of R and read S entirely in the second loop. So, the total number of disk I/O's is (1,000/100)(100+1,500)=16,000

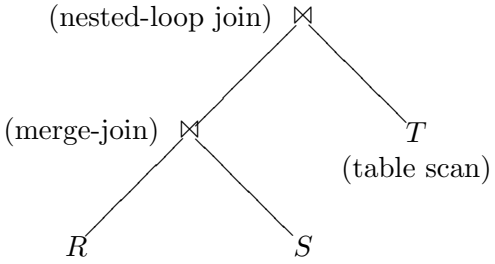
(v) The approach needs $3B(R) + 3B(S) = 75,000$. We have 25 subsorted lists for relations R and S. Therefore, $K \leq M - 25$ must holds.

Problem 5 (*17 points*)

(i)

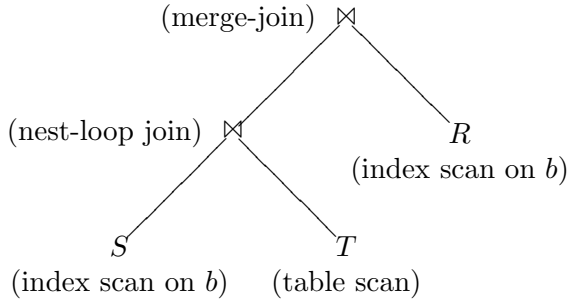






(index scan on b) (index scan on b)

(ii)



There may be several answers.

(iii)

Plan	$R \bowtie S$	interesting order?
Plan A	(table-scan R) nested-loop-join (table-scan S)	no
Plan B	(index-scan R on b) merge-join (index-scan S on b)	yes
Plan C	(index-scan R on b) nested-loop-join (table-scan S)	yes

Plan A: (table-scan R), (table-scan S) produce no order, and nested-loop-join does not produce any sorted order.

Plan B: (index-scan R on b) produces tuples sorted on b, (index-scan S on b) produces tuples sorted on b, and merge-join will produce a sorted output on b: Thus, it will produce an interesting order.

Plan C: (index-scan R on b) produces tuples sorted on b, and using nested-loop-join with a sorted relation as the outer loop will produce a sorted output: Thus, it will produce an interesting order.

(iv)

Plan	estimated cost	keep?
Plan A	2000	yes
Plan B	3000	yes
Plan C	4000	no

Plan A: The plan has the smallest cost, so it will be kept.

Plan B: It produces an interesting order, although does not have the smallest cost, so will be kept.

Plan C: Although it produces an interesting order, its cost is higher than Plan B, so this plan won't be kept.

Problem 6 (*12 points*)

(i) A= 21, B= 11 or 41, C = 30 or 31 or 32 or 33, D = 50, 51, 52

(ii) T1, T6.

(iii) T3, T4, T5

(iv) A = 20 B = 41 C = 31 D = 52
A= 20 because T1 is redone.
B= 41 because T4 is redone.

C= 31 because both T1 and T6 are undone and T3 is redone.
D = 52 because T5 is redone.

Problem 7 (*12 points*)

Answer the following questions:

(i) We do not need to shut down the system while the checkpointing is being made, so new transactions enter the system during checkpointing.

(ii) No, it is not since the graph has cycle.

(iii) Yes, deadlock occurs with such an interleaving of the actions of these transactions. T_1 can gain read lock on X and T_2 can gain read lock on Y, but T_2 cannot gain write lock on X because it has conflict with read lock of transaction T_1, so T_2 has to wait. Similarly, T_1 cannot gain write lock on Y due to the conflict of read lock of the transaction T_1. So, T_2 cannot proceed as well and they will wait forever.

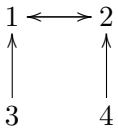
(iv) As T_1 is the first to validate, there is nothing to check; T_1 validates successfully. T_3 validates next. The only other validated transaction is T_1, and T_1 has not yet finished. Thus, both the read- and write-sets of T_3 must be compared with the write-set of T_1. However, T_1 writes only A, and T_3 neither reads nor writes A, so T_3's validation succeeds. Last, T_2 validates. Both T_1 and T_3 finish after T_2 started, so we must compare the read-set of T_2 with the write-sets of both T_1 and T_3. In addition, since T_3 has not finished yet when T_2 is validating, the write-set of T_2 must be compared with the write set of T_3. However, there is no common element in the two sets. Thus, T_2 can also validates.

Problem 8 (*10 points*)

(i)

$l_3(C)$ denied (C locked by 1)
$l_1(B)$ denied (B locked by 2)
$l_4(D)$ denied (D locked by 2)
$l_2(A)$ denied (A locked by 1)

wait-for graph is:



Yes, a deadlock occurs, because of a cycle between 1 and 2 in the wait-for graph.

(ii)

Yes, it can have a deadlock.

example:
T1 starts
T2 starts
T1 reads A
T1 writes A

T2 writes B
T1 writes B (result: T1 wait for T2 to commit or abort)
T2 reads A (result: T2 wait for T1 to commit or abort)
deadlock occurs

CSE 444 Midterm Exam

July 28, 2010

Name _____Sample Solution_____

Question 1	/ 28
Question 2	/ 20
Question 3	/ 16
Question 4	/ 20
Question 5	/ 16
Total	/ 100

The exam is open textbook and open lecture notes, including any marginal or other notes you have made during lecture. Please put away all other materials including projects, homework, old exams, and sample solutions. No computers, electronics, communications, or other devices are permitted.

Please wait to turn the page until everyone has their exam and you are told to begin.

Question 1. SQL (28 points, 7 each part) We have a small database with three tables to keep track of mountains and the people who climb them. The tables and their attributes are as follows:

Mountain(name, height, country, state)
Climber(cid, name, sex)
Ascent(cid, mountain, month, year)

- Mountain: name (string) is assumed to be unique for this problem; height in feet (integer); country and state (or province, district, etc.) where the mountain is located (strings)
- Climber: cid (unique integer), name (string), sex (string, either 'M' or 'F', and never null). Different climbers may have the same name, but they will have different cid numbers.
- Ascent: record of a single climb by a single climber. cid is a foreign key referencing Climber, mountain is a foreign key referencing a name in Mountain, month and year are integers. We assume that no climber climbs a single mountain more than once in a given month and year.

(a) Write a SQL query that gives the names of all climbers who have climbed the tallest mountain. You may assume that there is only one mountain that is taller than all of the others. If two or more climbers have the same name, it is your choice whether that name appears more than once.

```
select c.name
from climber c, ascent a, mountain m
where c.cid = a.cid and
    a.mountain = m.name and
    m.height >= all (select height from mountain)
```

(b) Write a SQL query that returns the percentage of women among the climbers that ascended the mountain named 'Rainier' in July 2007. You may assume there are at least one man and one woman in the list of climbers that ascended Rainier during that month.

```
select 100 * women.nbr / total.nbr
from    (select count(*) as nbr
         from ascent a, climber c
         where a.cid = c.cid and a.mountain = 'Rainier' and
         a.month = 7 and  a.year = 2007)
         as total,
         (select count(*) as nbr
         from ascent a, climber c
         where a.cid = c.cid and a.mountain = 'Rainier' and
         a.month = 7 and a.year = 2007 and c.sex = 'F')
         as women
```

(continued next page)

Question 1. (cont) Schemas repeated for convenience:

Mountain(name, height, country, state)
Climber(cid, name, sex)
Ascent(cid, mountain, month, year)

(c) Write a SQL query that returns a list of every mountain in the database and the number of different climbers who have ascended each mountain, sorted in descending order by number of climbers. Climbers may have climbed the same mountain many times, but each unique climber should only be counted once in the total for each mountain. If several mountains have the same number of climbers, those mountains with the same number may be listed in any order.

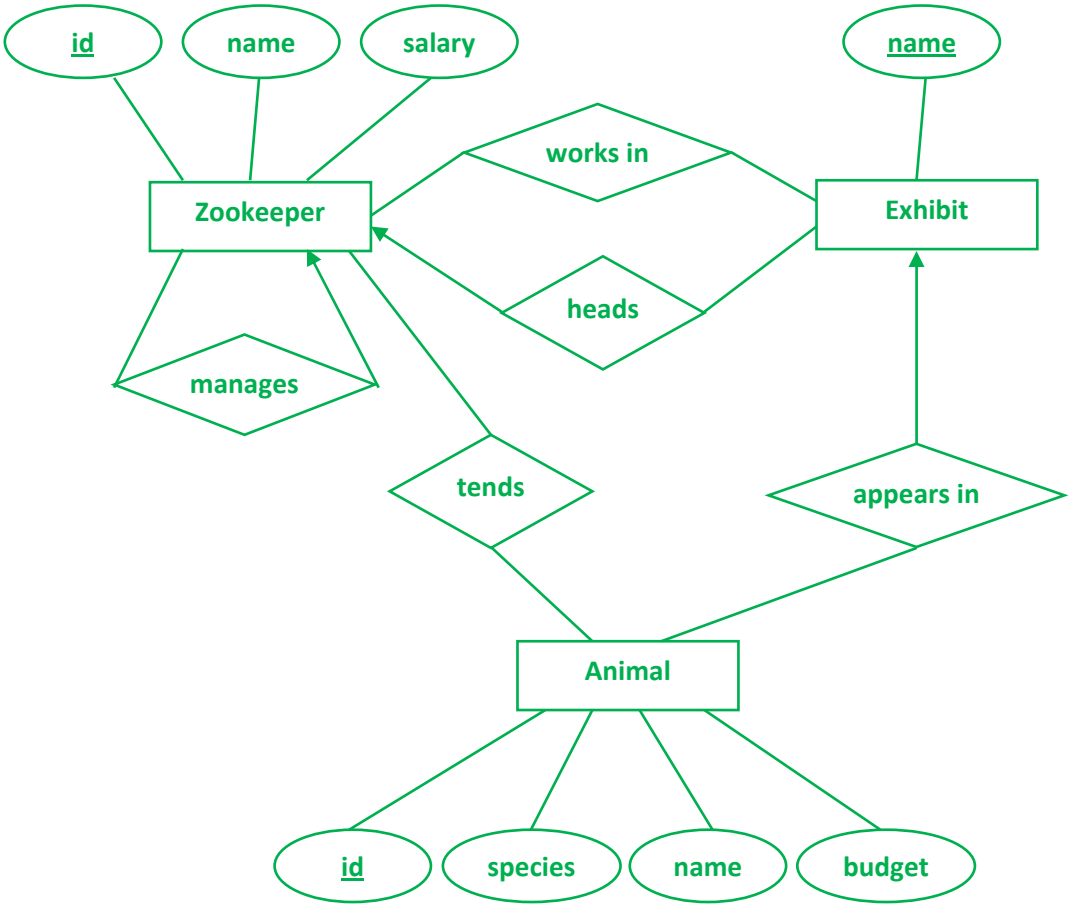
```
select count (distinct a.cid) as c_climbers
from mountain m left outer join ascent a
    on m.name = a.mountain
group by m.name
order by c_climbers desc;
```

(d) Write a SQL query that determines the state in the country 'USA' that has the most mountains at least 12,000 feet high, and returns the name of that state and a list of those mountains, sorted by mountain name. You may assume there is only one state with this "largest number of tall mountains". You may repeat the name of the state along with each mountain name in the output if that is more convenient.

```
select name, state
from mountain
where country = 'USA' and
   state = (
          select state
          from mountain
          where country = 'USA' and height >= 12000
          group by state
          having count(*) >= all (
                    select count(*)
                    from mountain
                    where country = 'USA' and height >= 12000
                    group by state
                    )
          )
   and height >= 12000
order by name;
```

Question 2. Conceptual design (20 points) We would like to design a database to keep track of a zoo.
(a) Give an E/R diagram that captures the following entities and relationships:

- The zoo employees are known as zookeepers. Each zookeeper has a name, a unique employee ID number, and a salary. He or she may work in any number of exhibits, or none at all (administrative staff do not work in exhibits, for example). A zookeeper does not have to work in an exhibit to head it.
- The zoo has several exhibits. Each one has an exhibit name and a head zookeeper who is in charge of it. A zookeeper may head many exhibits, but each exhibit has at most one head.
- Each zookeeper has at most one manager, who is another zookeeper (some zookeepers, such as the director of the zoo, have no manager). A zookeeper can manage any number of employees, including none.
- The zoo has many animals. Each has a name, species, unique ID number, and a weekly budget for its care. Each animal is tended by at least one zookeeper, and may appear in at most one exhibit. Some animals, such as those housed in the veterinary clinic, are not in any exhibit.



(continued next page)

Question 2 (cont.) (b) Give SQL CREATE TABLE statements for tables that implement your E/R diagram from part (a). You should give the attribute names and types for each table, and clearly indicate which attributes are keys, which others are unique, and which are foreign keys referencing other tables.

```
CREATE TABLE Zookeeper (
        id INTEGER PRIMARY KEY,
        name VARCHAR(100),
        salary_cents INTEGER,
        manager_id INTEGER REFERENCES Zookeeper(id)
);

CREATE TABLE Exhibit (
        name VARCHAR(100) PRIMARY KEY,                  -- PRIMARY KEY is optional here
        head_id INTEGER REFERENCES Zookeeper(id)
);

CREATE TABLE Animal (
        id INTEGER PRIMARY KEY,
        species VARCHAR(100),
        name VARCHAR(100),
        budget_cents INTEGER,
        appears_in VARCHR(100) REFERENCES Exhibit(name)
);

CREATE TABLE WorksIn (
        zkid INTEGER REFERENCES Zookeeper(id),
        exname VARCHAR(100) REFERENCES Exhibit(name),
        PRIMARY KEY(zkid, exname)
);

CREATE TABLE Tends (
        zkid INTEGER REFERENCES Zookeeper(id),
        aid INTEGER REFERENCES Animal(id),
        PRIMARY KEY(zkid, aid)
);
```

Note: If we could do it over again, it would have been better to just ask for the table schemas instead of the full CREATE TABLE SQL statements in order to cut down on the amount of writing needed.

Question 3. BCNF (16 points) Suppose we have a relational schema R(A,B,C,D,E) with the following functional dependencies:

- A -> E
- C -> D
- A,B -> C,D

Decompose this relation, if needed, into collections of relations that are in BCNF. At each step, show your work and explain which dependency violation(s) you are correcting. Be sure the steps in the decomposition are clear and that it is clear which tables are the final ones. Also, identify the keys of each table by underlying the attribute(s) that make up the key.

Hint: there may be more than one correct solution to the problem.

The closures of the various dependencies are:

{AB}+ = {ABCDE}
A+ = {AE}
C+ = {CD}

The first one is not a problem, the other two violate BCNF.

Solution I. Use A->E to decompose the table into

R1(A,E)
R2(A,B,C,D)

R1 is in BCNF. R2 has a bad dependency C->D. Use that to decompose and we get

R21(C,D)
R22(A,B,C)

The final tables are R1, R21, and R22, which are in BCNF.

Solution II. If we use C->D for the first decomposition we get

R1(C,D)
R2(A,B,C,E)

R1 is in BCNF, R2 has a bad dependency A->E. Decompose R2 to get

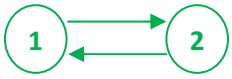
R21(A,E)
R22(A,B,C)

The final tables are R1, R21, and R22.

Question 4. Serialization (20 points). For each of the following schedules,

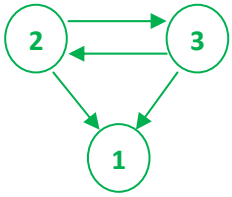
i. Draw the precedence graph for the schedule.
ii. If the schedule is conflict-serializable, give the equivalent serial schedule. If the schedule is not conflict-serializable, explain why not.
iii. If the schedule is not conflict-serializable, but there still is an equivalent serial schedule, give that schedule and explain why it is equivalent.

(a) r2(X) r1(X) r2(Z) w2(Z) w2(X) r1(Z) w1(X)



This is not conflict-serializable because there is a cycle in the precedence graph. Further, there is no equivalent serial schedule, since T1 must read X before T2 writes it, but T2 must write Z before T1 reads it.

(b) r2(X) r3(Y) w3(X) r1(Y) w2(X) w1(Y) w1(X)



This is not conflict serializable because there is a cycle in the graph.

However, the schedule r2(X) w2(X) r3(Y) w3(X) r1(Y) w1(Y) w1(X) is serial and is view equivalent to the original schedule. The writes to X by T2 and T3 are superseded by w1(X) at the end of the schedule before any other transactions read X.

Question 5. (16 points) Assorted short questions.

(a) In a system with a simple **undo** log, when a transaction wants to commit it must first wait until all of its data pages have been written to disk. True or false? ___**true**_____

(b) In a system with a simple **redo** log, changes to the transactions data pages can be written before or after they are written to the log. True or false? ___**false**_____
(In a redo log, outputs must be done after they are logged)

(c) in a system with a simple **redo** log using non-quiescent checkpointing, an <END CKPT> record in the log indicates that all transactions identified in the corresponding <START CKPT> record have either committed or aborted. True or false? ___**false**_____
(END CKPT only indicates that all dirty pages belonging to transactions that committed before the <START CKPT> are now on disk)

(d) In the ARIES recovery protocol, after a crash, the analysis pass examines the log and determines two things. One is the list of active transactions that did not finish at the time of the crash and will need to be undone. The other is a number known as FirstLSN that identifies an entry (LSN) in the log. What is the significance of this number?

This identifies the earliest entry in the log that must be examined on the redo pass. (i.e., the first log entry whose changes to the database might have been lost in the crash)

(e) In the ARIES recovery protocol, if a crash occurs during the undo phase of the recovery, then after a restart, all of the undo operations from before the crash will be repeated because they are idempotent.
True or false? ___**false**_____
(Each ARIES undo operation is logged with a CLR. The CLR will be redone if there is a crash before the undo phase finishes; no undo operation happens more than once.)

(f) In a system using the ARIES recovery protocol, changes made to transaction data pages may be written to disk before or after the transaction's commit record is written to the log.
True or false? ___**true**_____

(g) In a scheduler that uses **timestamps** for concurrency control, each database element X has a read timestamp RT(X). Whenever a transaction reads a database element X, the element's RT(X) timestamp is updated by recording the transaction's timestamp in RT(X) for that database element.
True or false? ___**false**_____
(RT(X) = max(old RT(X) value, timestamp on current transaction))

(h) In a scheduler that uses **validation** for concurrency control, the committed transactions have the same effect as they would in a serial schedule where the transactions are executed in the order in which they started (i.e., transaction start times determine the serialization order).
True or false? ___**false**_____
(Serialization order = transaction validation order)