Name:_____ **Student ID:**_____

MIDTERM EXAM, Fall 2009

CS 564
Department of Computer Science
University of Wisconsin, Madison

Exam Rules:

1) Close book and notes, 75 minutes

2) Please write down your name and student ID number NOW.

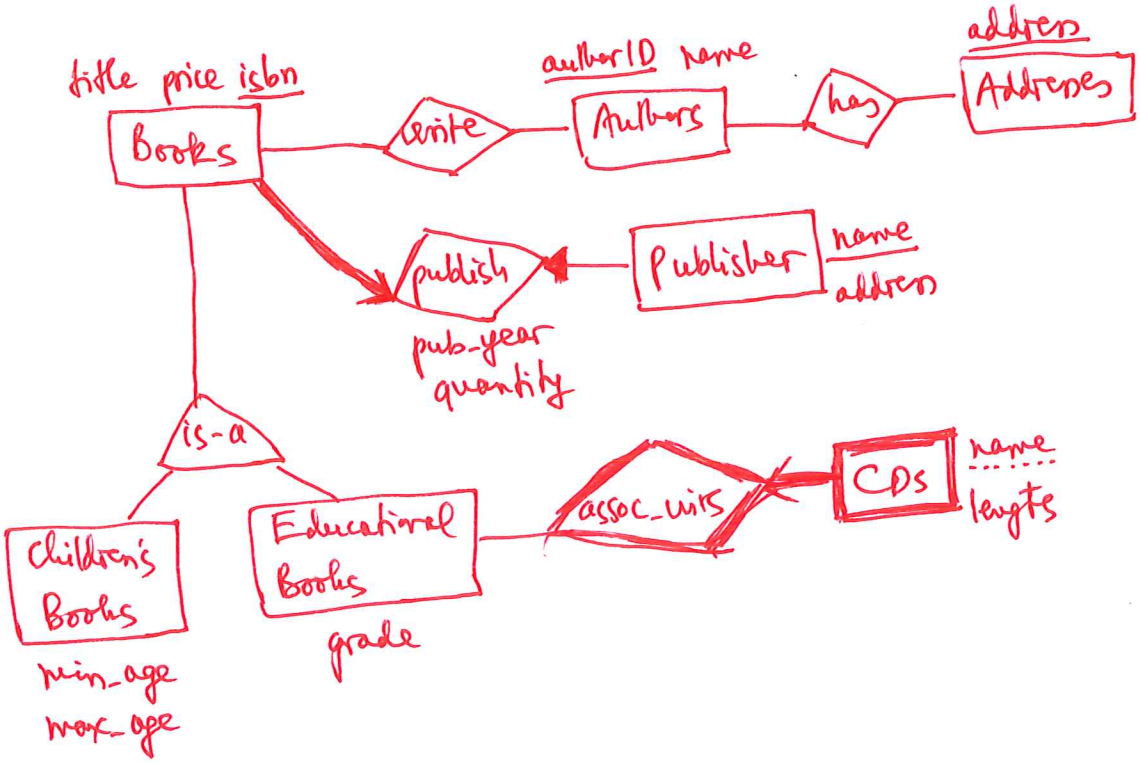
3) Please wait until being told to start reading and working on the exam.

4) If you think a problem is ambiguous, write down your assumptions, argue that they are reasonable, then work on the problem using those assumptions.

1. (20 points) Suppose that you are asked to write a database application that captures the following information:

We want to store information about books. Each book has a title, a price, and an ISBN number. The ISBN number uniquely identifies the book. Some books are children's books. Each children's book also specifies the minimum age and the maximum age, which together indicate the ideal age range for its readers. Some books are educational books. Each educational book also specifies the grade, that is, the school grade that the book is most suited for. Each educational book may also be associated with one or multiple exercise CDs. Each CD has a name and a length (in minutes). We can only uniquely identify each CD using its name and the information of the book that the CD is associated with. Each book is written by one or many authors. Each author has an author ID, a name, and multiple addresses. Each book is also published by exactly one publisher. Each publisher has a name (which uniquely identifies the publisher) and a single address. When a publisher publishes a book, we want to record the publishing year and the quantity (that is, the number of copies that the publisher produces).

a) [10 points] Draw an ER diagram for this application. Decide the key attributes and identify those.



b) [10 points] Convert the above ER diagram into a relational schema. Specify the primary key of each relation in your schema.

Books(isbn, title, price, pub-name, pub-addres, pub-year, quantity)

ChildrenBooks(isbn, minage, maxage)

EduBooks (isbn, grade)

CDs (isbn, cd-name, lengts)

Authors(authorID, name) note: primary key is just authorID

Addresses(authorID, address)

Write(isbn, authorID)

2. (30 points; 10 points each)
Consider a database schema with the following relations:
Suppliers(sid: integer, sname: string, address: string)
Parts(pid: integer, pname:string, color: string)
Catalog(sid:integer, pid:integer, cost:real)

Please write the following SQL queries:

(a) Find the sids of suppliers who sell a 'red' part that costs more than 100. Your query should not return duplicates.

Select distinct C.sid
From Parts P, Catalog C
where (C.pid = P.pid) ∧ (color = red) ∧ (cost > 100)

(b) An expensive part is a part that has cost > 100 from some supplier. Find the sids of those suppliers who sell at least one such part.

Select C.sid
From Catalog C
Where cost > 100

(c) Continuing from Part b, for each supplier that sells at least one expensive part, return the sid and the total number of expensive parts that that supplier sells.

```
Select sid, count(*) as pcount
From Catalog C
Where cost >100
groupby sid
```

3. (10 points) Briefly describe sequential flooding and explain how an RDBMS avoids sequential flooding.

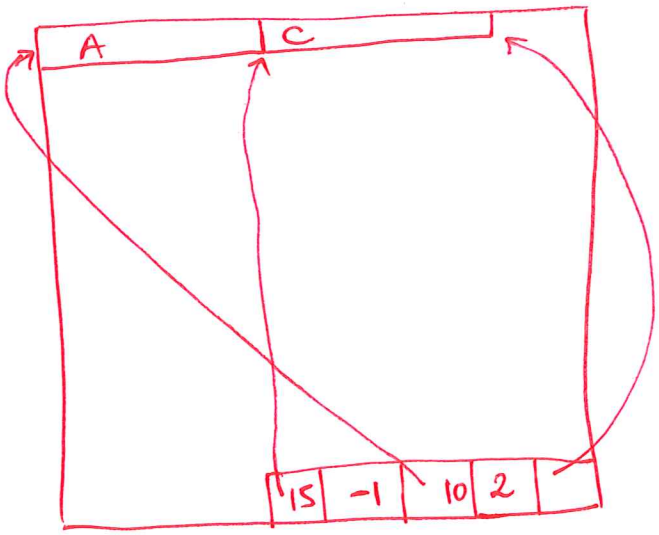
See textbook.

4. (20 points, 10 each)

a) Consider slotted pages that are used to store variable-length records. Assume a newly created page P. Consider the following sequence of actions:

Insert record A of length 10 bytes. Insert record B of length 20. Insert record C of length 15. Insert record D of length 25. Delete record B. Delete record D. Compact the page (that is, move all records to be contiguous to one another, starting from the beginning of the page, to free up space for new records).

Draw the page after the above sequence of actions has been executed.



Here -1 is used to mark an unused slot.

b) As discussed in the class, fixed-length records can be stored using either (1) the slotted page for variable-length records (as mentioned in Part a), or (2) the unpacked, bitmap page for fixed-length record. In Option (2), the page is divided into slots. We use an array of bits, one per slot, to keep track of free slot information. Locating records on the page requires scanning the bit array to find slots whose bits are on. When we delete a record, its bit is turned off.

Under what conditions would it be better to store fixed-length records using Option (2) instead of Option (1)?

If we don't need to move records on a page around frequently (for example, we don't have to keep them sorted), and if we do a lot of record lookup, then option 2 is better than option 1. In this case, finding a record incurs less overhead. For example, to find the record with id (3,5), we go directly to slot #5 of page #3.

5. (20 points + 5 point bonus): Recall that each internal page of a B+ tree contains search keys (e.g., 23, 45, etc.) and page ids (i.e., pointers to lower-level pages). Suppose a page stores 1004 bytes, a search key is 6 bytes, and a page id is 4 bytes. Suppose there are no duplicate search keys.

(a) [10 points] Compute the degree of the tree such that each internal page is utilized to the fullest extent (that is, we can store the maximal number of search keys on each page).

Let d be the degree of the tree. Then each page stores 2d search keys and (2d+1) pointers (that is, page IDs).

So $2d * 6 + (2d+1) * 4 = 1004$

So $d = 50$.

(b) [10 points] Suppose we have a non-clustered B+ tree index over 400,000 tuples. Suppose further that a record id is also 4 bytes. Finally, suppose that each internal page is 70% full (that is, if the tree degree is d, then each internal page contains roughly 70% * 2d pointers). How many memory pages do we need, at the bare minimum, to store the above B+ tree index so that any equality search will take at most 2 I/Os?

① First, compute how many leaf pages of the tree do we need to be able to index 400,000 tuples.

~~Each leaf page~~ To index a tuple, we need a search key (6 bytes) and a record id (4 bytes), so we need 10 bytes. So a single leaf page can index 1004 / 10 = 100 tuples. So we need 400,000 / 100 = 4000 leaf pages for the tree.

② Next, we figure out how many levels the tree must have to have at least 4000 leaf pages. The branch out factor is 70% * 2 * 50 = 70. So we will need a tree of 3 levels, with 1 page for root, 70 pages at level 2, and 70 × 70 = 4900 pages at level 3.

③ Now it is clear that we need at least 1+70 = 71 pages of memory, to store levels 1 and 2 of the tree. This way, any equality searches will require reading at worst a page at level 3 and a data page, for at worst 2 I/Os.

(c) [BONUS QUESTION. 5 POINTS. YOU DON'T HAVE TO SOLVE THIS TO GET THE FULL 100 POINTS]: suppose each internal page is F% full. Show the equations you need to solve in order to find out how many memory pages we need, at the bare minimum, to store the B+ tree in Part b, so that any equality search will take at most 3 I/Os?