CS 564 Midterm Exam

Open book, open notes, open internet – closed friends and neighbors

If you find a question ambiguous or difficult, please do not ask other students in the class. Instead, please post your questions on Piazza. Please do not post answers to other students' questions – please leave that to the instructor and the TAs. Feel free to post follow-up questions if you find an answer ambiguous. Before you post a question, please check whether a similar question is already posted and perhaps even answered.

Please submit your exam as a single pdf file by **Friday, November 1, 12:00PM noon** (not midnight) on Canvas. Make sure to submit before the deadline-- late submissions will not be accepted.

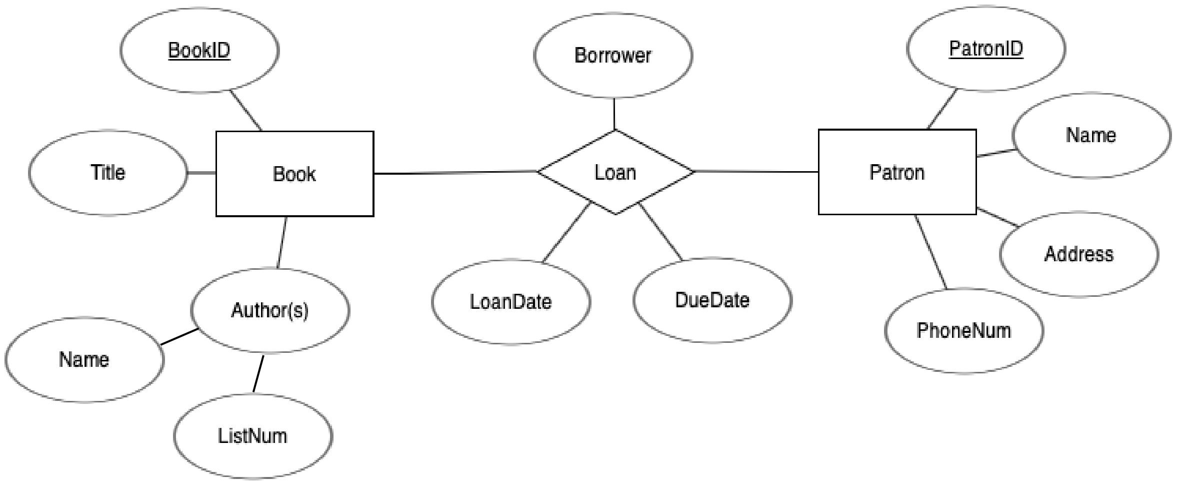
Unless you have very readable handwriting, please type your answers. If we can't read it, we can't credit it. For some questions, please draw by hand and label by hand, but make sure it's all readable – then scan it or photograph it and include it in your one pdf file.
For questions 8-11, please be concise and use short, succinct sentences. 30-100 words should suffice. Do not copy from the book, from Wikipedia, or from other internet sources.

Starting on the next page, answer each of the following questions or assignments:

Q1. Draw an ER diagram for a (much simplified) database for a lending library. Map the following concepts to entities, relationships, and attributes: book, title, author(s), borrower, loan date (when the book was borrowed from the library), due date, patron (a person who may borrow books), name, address, phone (some attributes apply to both the library and to each patron). You may add a few attributes but keep that to a small number. For better or worse, there may be books with the same title, but perhaps by different authors. A patron may borrow the same book multiple times.

A1:

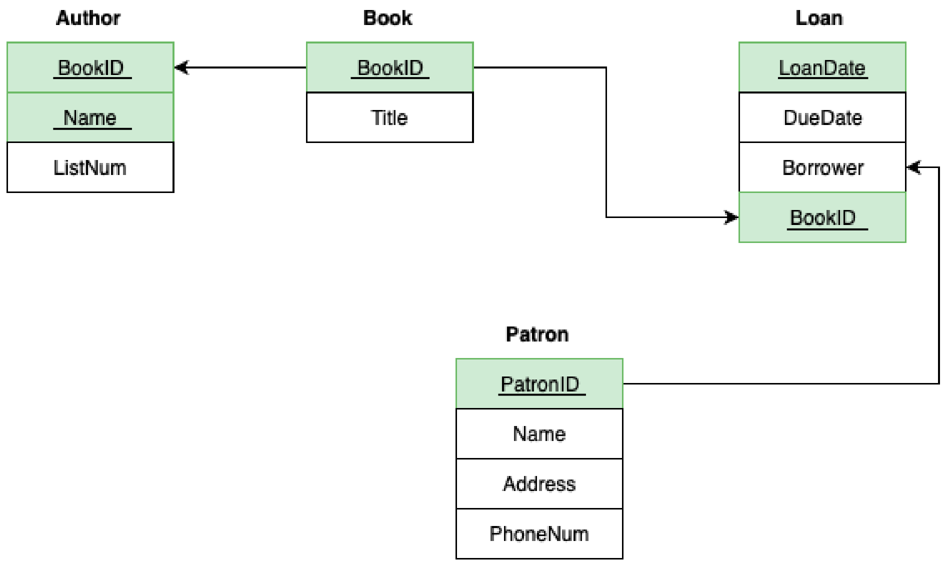
- The rectangular boxes are entities
- The elliptocytosis boxes are attributes
- The rhombus are relationships
- Added "BookID" and "PatronID" as additional attributes



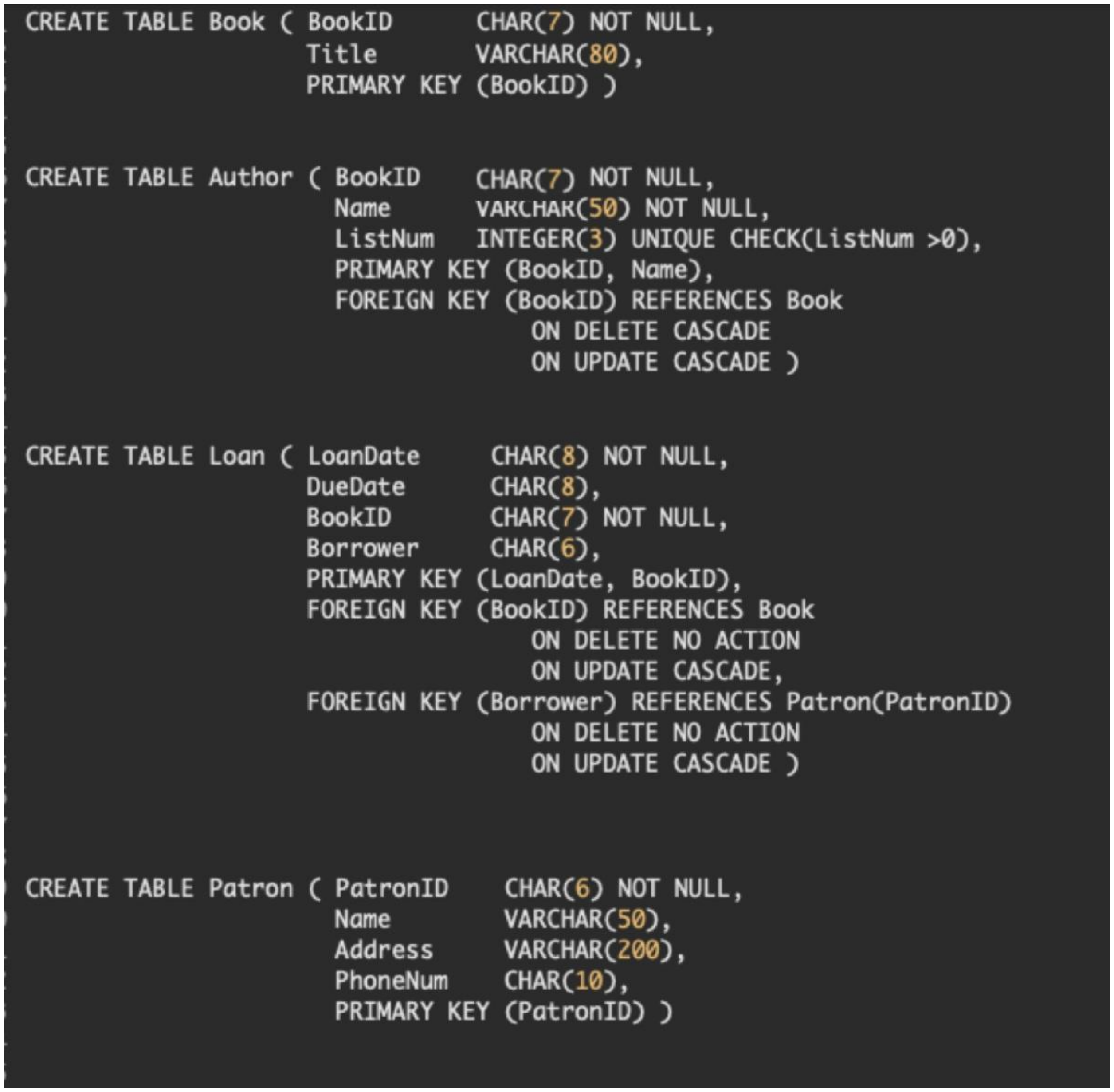
Q2. Map your ER diagram to a schema (table names, column names, primary and foreign keys). You can use the TPC-H schema as an example, but make sure you clearly indicate the primary and foreign keys.

A2:

- The green boxes are indicating the primary keys of the tables
- The arrows are pointing to the foreign keys of the tables
- In the Loan Table, Borrower and BookID are foreign keys
- In the Author Table, BookID is the foreign key



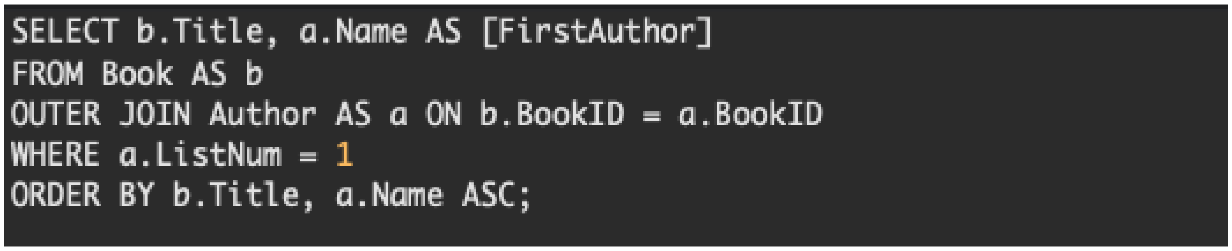
Q3. Write “create table” statements for these tables – include all integrity constraints that make sense. Try to include each kind of integrity constraint at least once in the database. Each table should have a primary key and each table should include a foreign key or be referenced by a foreign key.



```
CREATE TABLE Book ( BookID      CHAR(7) NOT NULL,
                    Title       VARCHAR(80),
                    PRIMARY KEY (BookID) )


CREATE TABLE Author ( BookID    CHAR(7) NOT NULL,
                      Name      VARCHAR(50) NOT NULL,
                      ListNum   INTEGER(3) UNIQUE CHECK(ListNum >0),
                      PRIMARY KEY (BookID, Name),
                      FOREIGN KEY (BookID) REFERENCES Book
                              ON DELETE CASCADE
                              ON UPDATE CASCADE )


CREATE TABLE Loan ( LoanDate     CHAR(8) NOT NULL,
                    DueDate      CHAR(8),
                    BookID       CHAR(7) NOT NULL,
                    Borrower     CHAR(6),
                    PRIMARY KEY (LoanDate, BookID),
                    FOREIGN KEY (BookID) REFERENCES Book
                              ON DELETE NO ACTION
                              ON UPDATE CASCADE,
                    FOREIGN KEY (Borrower) REFERENCES Patron(PatronID)
                              ON DELETE NO ACTION
                              ON UPDATE CASCADE )



CREATE TABLE Patron ( PatronID    CHAR(6) NOT NULL,
                      Name        VARCHAR(50),
                      Address     VARCHAR(200),
                      PhoneNum    CHAR(10),
                      PRIMARY KEY (PatronID) )
```

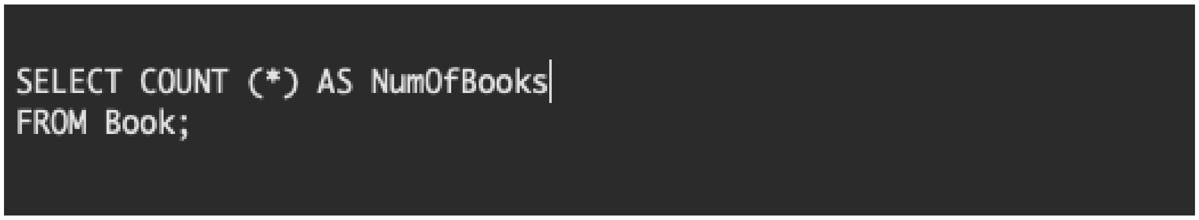
Q4. Write the following queries over your tables. Feel free to use “with” clauses where convenient; use nested queries sparingly.

a. List all book titles in ascending order – include the first author’s name for each book and include duplicate titles.



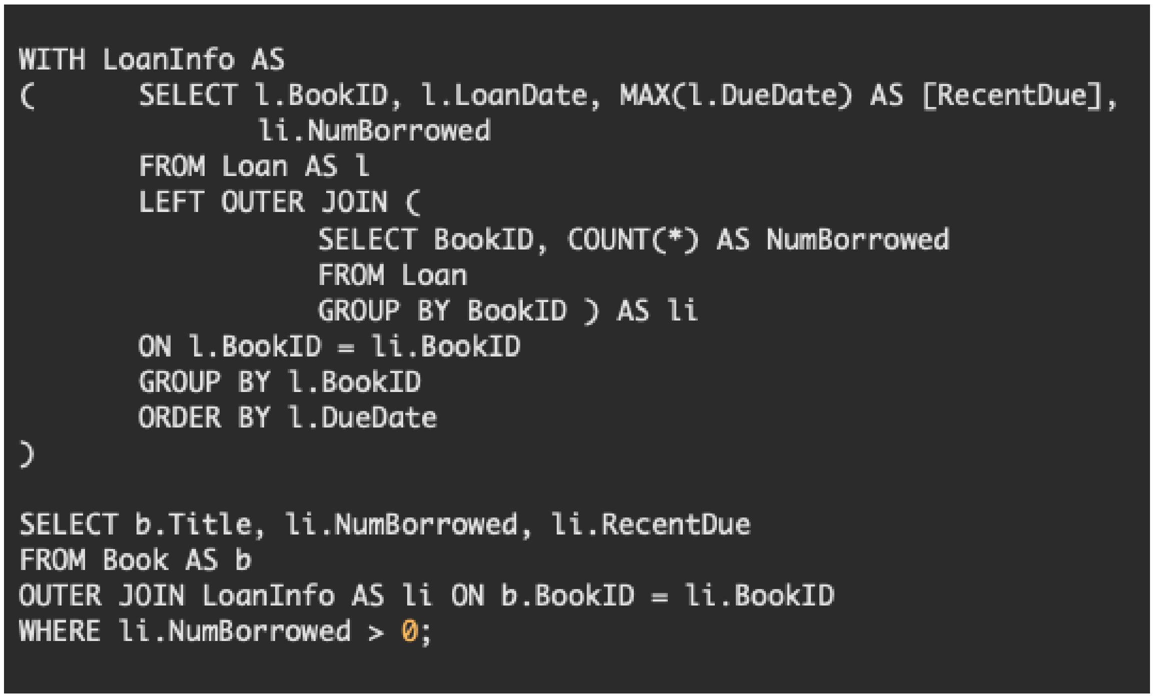
```
SELECT b.Title, a.Name AS [FirstAuthor]
FROM Book AS b
OUTER JOIN Author AS a ON b.BookID = a.BookID
WHERE a.ListNum = 1
ORDER BY b.Title, a.Name ASC;
```

b. Count the number of books that the library owns.



```
SELECT COUNT (*) AS NumOfBooks
FROM Book;
```

c. For each book borrowed at least once, list the title, the number of times it has been borrowed, and the most recent due date (whether that due date is in the past or in the future).

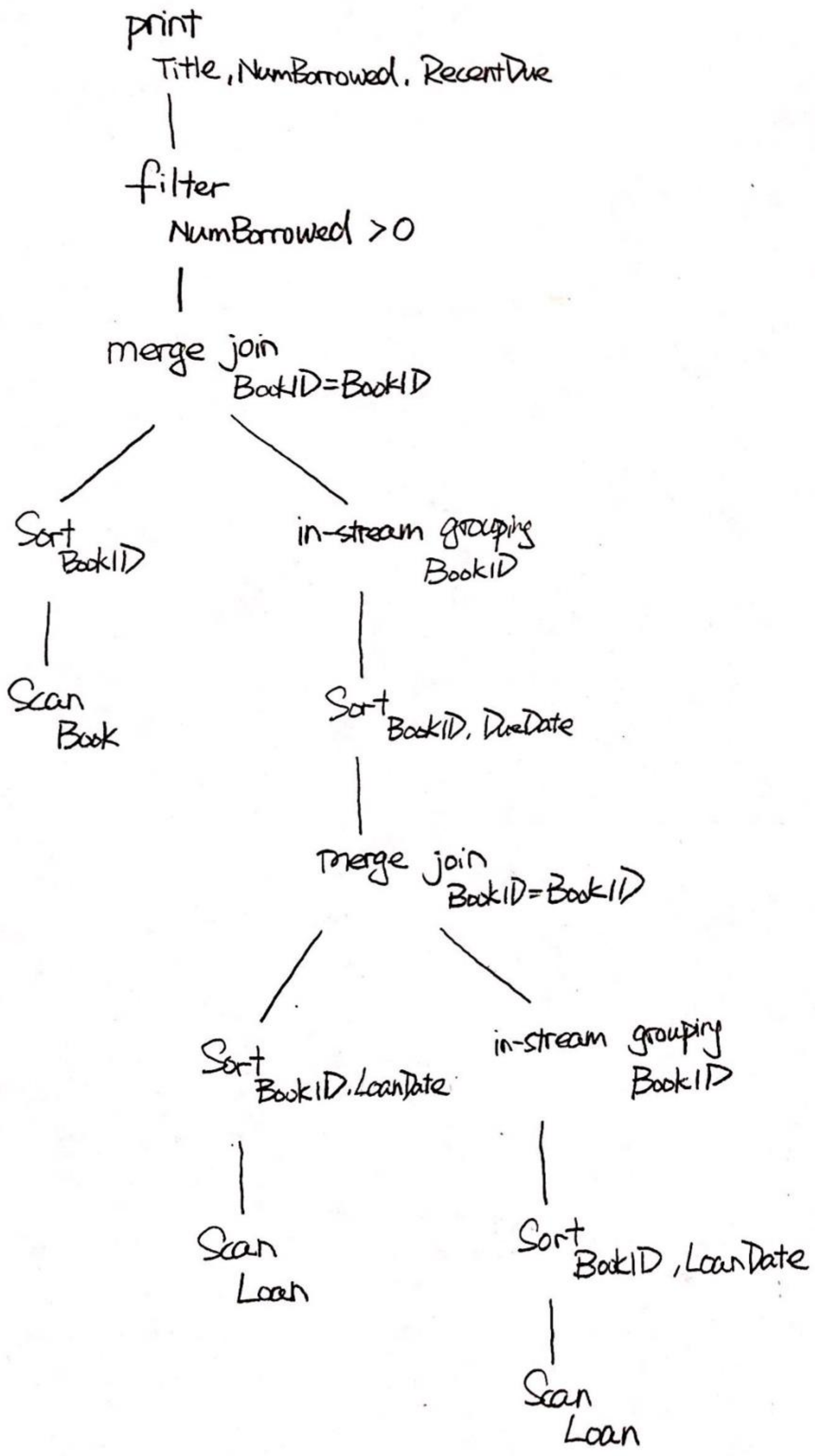


```
WITH LoanInfo AS
(       SELECT l.BookID, l.LoanDate, MAX(l.DueDate) AS [RecentDue],
                li.NumBorrowed
        FROM Loan AS l
        LEFT OUTER JOIN (
                    SELECT BookID, COUNT(*) AS NumBorrowed
                    FROM Loan
                    GROUP BY BookID ) AS li
        ON l.BookID = li.BookID
        GROUP BY l.BookID
        ORDER BY l.DueDate
)

SELECT b.Title, li.NumBorrowed, li.RecentDue
FROM Book AS b
OUTER JOIN LoanInfo AS li ON b.BookID = li.BookID
WHERE li.NumBorrowed > 0;
```

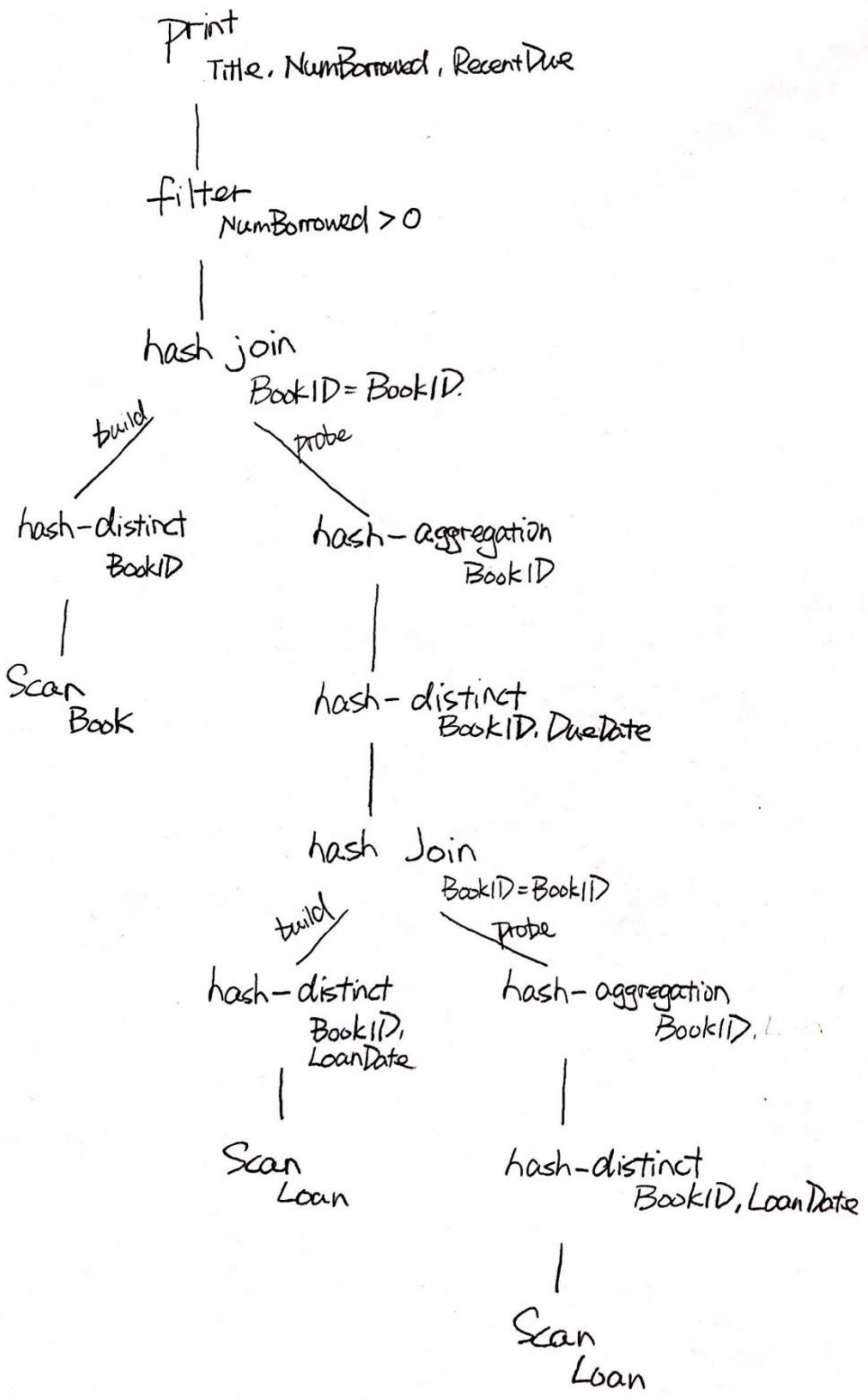
Q5. For the last query in question 4, draw or write an expression in relational algebra. (As we did in class, assume the algebra includes a grouping/aggregation operation in addition to the usual select, project, and join operations).

$$\begin{tikzcd}
 & \pi_{\text{Title, NumBorrowed. RecentDue}} \arrow[d, dash] & & \\
 & \sigma_{\text{NumBorrowed} > 0} \arrow[d, dash] & & \\
 & \bowtie_{\text{BookID}} \arrow[dl, dash] \arrow[dr, dash] & & \\
\text{Book} & & \text{LoanInfo} \arrow[d, dash] & \\
 & & \pi_{\text{BookID, LoanDate. RecentDue, NumBorrowed}} \arrow[d, dash] & \\
 & & \gamma_{\text{MAX(DueDate)}} \arrow[d, dash] & \\
 & & \text{Group By BookID.} \arrow[d, dash] & \\
 & & \bowtie_{\text{BookID}} \arrow[dl, dash] \arrow[dr, dash] & \\
 & \text{Loan} & & \pi_{\text{NumBorrowed, BookID}} \arrow[d, dash] \\
 & & & \gamma_{\text{COUNT(*)}} \arrow[d, dash] \\
 & & & \text{Group By BookID} \arrow[d, dash] \\
 & & & \text{Loan}
\end{tikzcd}$$

Q6. Map your relational algebra expression to a query execution plan (draw or write).



Q7. Provide an alternative query execution plan (draw or write).



Q8. Decide which one of your two alternatives is likely more efficient – state your choice and give reasons for your choice, or state that you can't decide and give reasons why either plan might be more efficient.

A8: The first plan used merge joins, and the second plan used hash joins. The second plan is likely more efficient using hash joins.
The Book table and Loan table could be very possibly having a very large amount of data of books and loaning information. The merge join required sorted inputs from both tables, and we have to do the sorting again to make sure the inputs are sorted as we wanted before every hash join. But the hash joins do not require sorted inputs, and we just need to perform the hash-distinct duplicates removal for the expected result before doing the hash join.
Although the merge join generally used less CPU and memory than the hash join, but the repeat sorting and merging such large amount of data do not give the merge join much advantage on memory used. Hence, the hash join plan is likely more efficient.

Q9. Explain (in your own words) the differences between an inner join, a (left) semi join, and a (left) outer join. Your explanation should address the sets of columns and the sets of rows in each operation's output.

A9:
Inner join: Inner join joins the tables with the given common columns as the condition to find and combine all the matching rows that exist in both tables. It focuses on joining the rows that have matched data in the given columns, other not matching rows won't show.

(Left) semi join (Half join): Semi join is not really combining the two tables, and it is more like a selection instead of a join. It returns the rows from left table that matched at least once by the given conditional columns with the subquery in EXISTS clause. Not allow duplicate row from the left table.

(Left) outer join: Outer join joins the tables and combines all the matching rows, but it also returns the unmatched rows from both tables that are joining. For left outer join, the unmatched rows in the left table still returned with some null values in the unmatched columns. It is not just focusing on joining the matched rows but also the unmatched rows from the indicated table.

Q10. Explain (in your own words) the differences between index nested loops join, merge join, and hash join (all used to compute an inner join). Your explanation should include required properties of the join inputs and properties of the join output. For the hash join, you may assume that one join input fits in memory. For extra points, explain what happens if neither join input fits in memory.

A10:
Index nested loops join: It is joining tables by using the nested loops (ex. for loops). The nested loops are for index searching

- No required inputs, both sorted and unsorted input are accepted, duplicates allowed
- Not really have properties, sorted or unsorted, duplicates allowed
- It is not directly needing the inputs to fit in memory, a much slower performance or even fail joining can happen if no input fits in.

Merge join: It is joining the two sorted table by index matching, and it needs at least one equi join condition to perform the join.

- Required Sorted input from both tables
- The output removes duplicates between two joined table but not duplicates within either of the table, sorted
- It is not directly needing the inputs to fit in memory, a much slower performance or even fail joining can happen if no input fits in.

Hash join: It joins the tables by hashing. There are two phases, the Build Phase and the Probe Phase. The hash function spills inputs into hash buckets in the build phase. It required at least one equi join condition to work.

- Good for large amount of inputs, both unsorted and unsorted inputs are accepted, duplicates allowed
- Unsorted output, duplicates might happen but can do duplicate removal
- It needs join input fits in memory, we will use Grace Hash Join if neither join input fits in memory. The Grace Hash Join perform partitioning and the joining.

Q11. For extra points, explain (in your own words) the differences and similarities between hash aggregation and hash join. For the hash join, you may assume that one join input fits in memory. For extra points, explain differences and similarities if neither input nor output fit in memory.

A11:
Differences: Hash Join is a joining operation, it purposes to join two tables by hashing. But the hash aggregation is a grouping operation, it is used for grouping the data with given condition in one table by hashing.

- If neither input nor output fit in memory, the hash join does the grace hash join which perform joining after partitioning., but the hash aggregation perform rehash after partition

Similarities: Both of their algorithm perform by hashing. They all use a hash function to split input tuples, and they all apply hashing to operate the data. Both of them not required sorted input, and they need memory and use blocking.

- If neither input nor output fit in memory, both hash join and hash aggregation do partitioning to spill the inputs to the smaller buckets first.