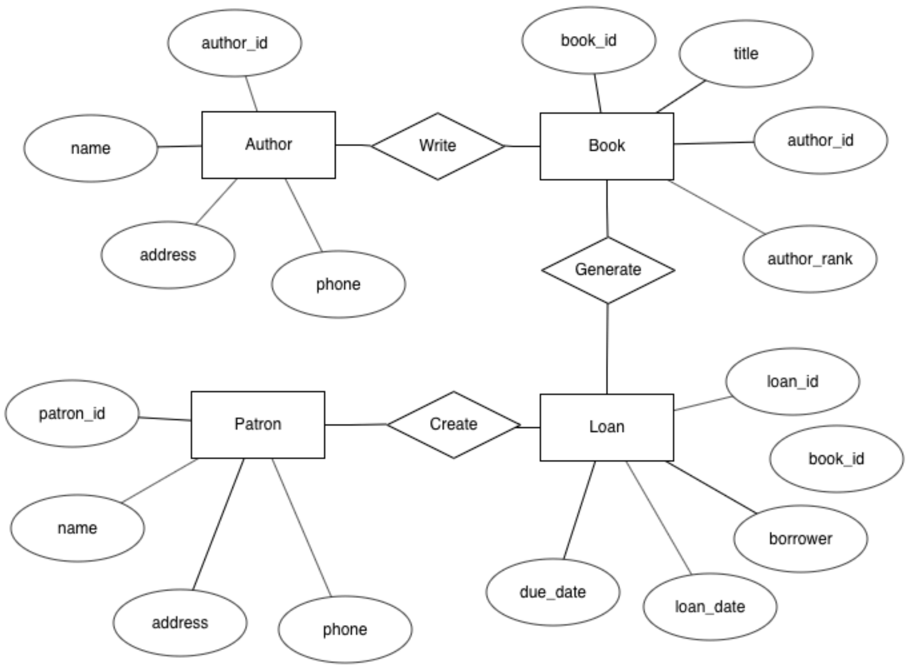
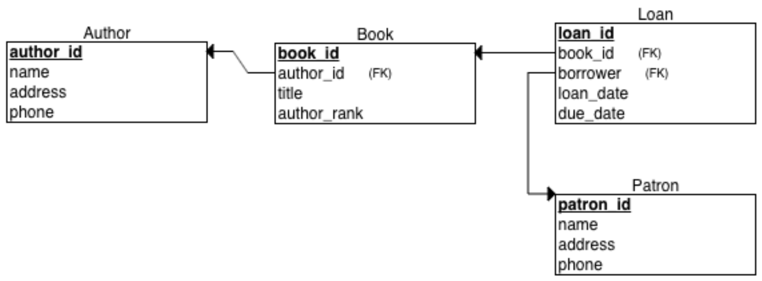
1.



2.



3. CREATE TABLE Author (author_id INTEGER, name CHAR(20), address CHAR(50), phone CHAR(50), PRIMARY KEYA(author_id));

   CREATE TABLE Book (book_id INTEGER, title CHAR(50), author_rank INTEGER, PRIMARY KEYA(book_id),
   FOREIGN KEY (author_id) REFERENCES Author (author_id));

   CREATE TABLE Patron (patron_id INTEGER, name CHAR(20), address CHAR(50), phone CHAR(50), PRIMARY KEYA(patron_id));

   CREATE TABLE Loan (loan_id INTEGER, loan_date datetime, due_date datetime,
   FOREIGN KEY (book_id) REFERENCES Book (book_id),
   FOREIGN KEY (borrower) REFERENCES Patron (patron_id));

4. (1)

```
  SELECT Book.title, Author.name
   FROM Book
    JOIN Author
      ON Book.author_id = Author.author_id
    AND Book.author_rank = 1
ORDER BY Book.title ASC
```

(2)

```
 SELECT COUNT ( DISTINCT book_id ) AS "Number of Books"
FROM Book;
```

(3)

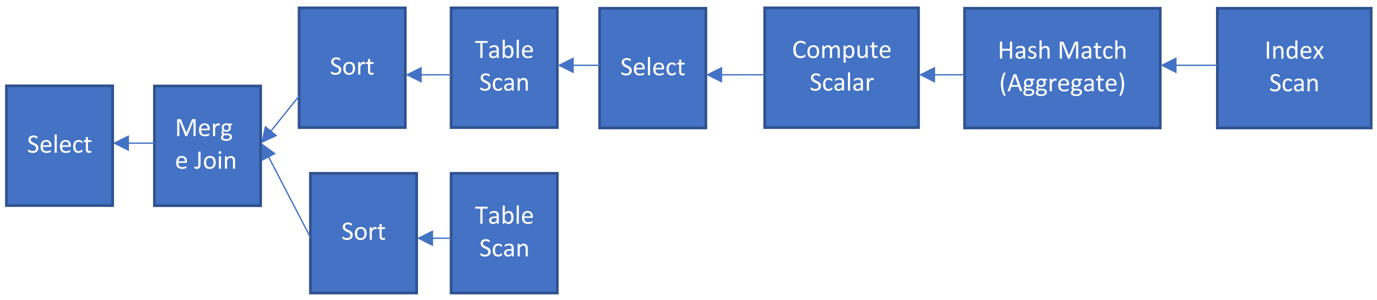
For each book borrowed at least once, list the title, the number of times it has been borrowed, and the most recent due date (whether that due date is in the past or in the future).

```
 WITH u AS (
 SELECT book_id, COUNT (loan_id), MAX (due_date)
  FROM Loan
GROUP BY book_id)
 SELECT Book.name, u. loan_id, u. due_date
  FROM u
   JOIN Book
     ON u.book_id=Book.book_id
```

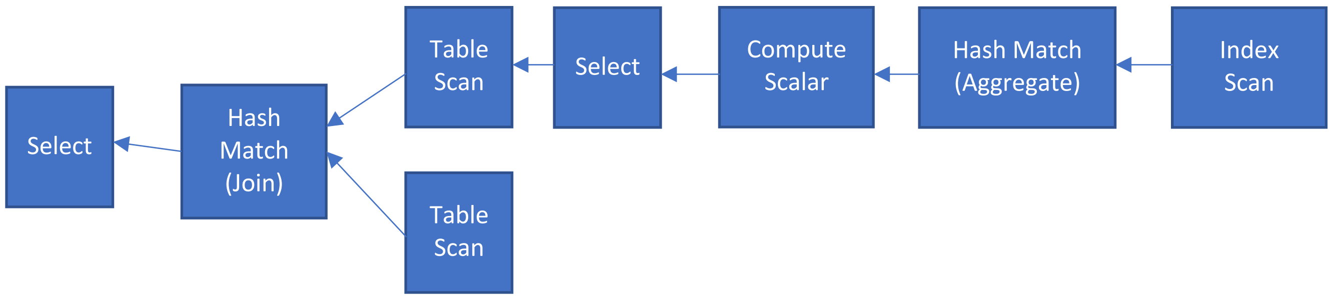
5.

$$\pi_{Book.name,u.loan_{id},u.due_{date}}\left(\rho_u\left({}_{book_{id}}F_{book_{id},count(loan_{id}),\max(due_{date})}(\text{Loan})\right)\infty_{u.book_{id}=Book.book_{id}} Book \right)$$

6.



7.



8.
If the book_id field in both tables are already sorted, choose merge join, otherwise hash join.

9.
INNER JOIN returns matched rows and all columns from both the first and second table .
A LEFT SEMI JOIN returns matched rows and columns from the first table.
LEFT OUTER JOIN returns all the rows and columns from first table and matched rows from second table.

10.
in Nested loops join, the secondary table is “driven” by the driving table. It scans the driving table first, and the result is matched against the secondary table. It requires a small driving table and small returned results, and it supports all joining conditions.

In Hash Join, the smaller table is used as driving table and fully scanned, and a RAM hash table is thus created. This hash table is a then matched with rows in the other table. It requires an equijoin predicate. The first-time joined result is returned slower due to hashing in memory.

Merge Join requires all input data to be sorted by the join columns, Then the Merge Join operator

For Hash Join, if neither join input fits in memory, the optimizer will split the fill into several files. Those cannot be fed into RAM will be stored on the disk temporarily.

11.
With a hash aggregate, we store one row for each group, so the total memory requirement is actually proportional to the number and size of the output groups or rows. With a hash join, we store each build row, so the total memory requirement is proportional to the number and size of the build rows.