1.) **Normalization Steps:**

Perform the following normalization steps on the given schema:

Step 1: **First Normal Form (1NF)**
Explain the 1NF rule and modify the schema to comply with it.

Answer:
For the initial schema given below, all the tables are already in First Normal Form since:

Atomicity – There are no columns containing multiple values or columns containing data that can be parsed. Each item based on the DB's logic couldn't further be broken down into sub values hence it being atomic.

Uniqueness – The table also has a clear primary key (BookID) which allows every row in the table to be unique.

No Derived or Calculated Data – Derived data are data that could be acquired through performing a query like the number of times a specific book was ordered. This can be calculated so we don't have to create a separate column keeping track of this data. Doing this can make our insert queries complicated because we then would have to keep updating every time we insert.

No Repeating Groups – Our table also do not have columns attempting to store multiple instances of a similar type of data. Like for the Book table, if it has multiple authors, we're not creating columns like Author1, Author2, etc. so it should be fine.

Step 2: **Second Normal Form (2NF)**
Explain the 2NF rule and modify the schema to comply with it.

Books Table
The Books Table is already in the second normal form. If we solely use data as the basis, BookID, Title, and Year can be our candidate keys. Since we have no column left to try and form a composite key with Author, it's safe to say that there are no partial dependencies here.

Customers Table
For the same reason with Books Table, the Customers table is already in the second normal form because based on the data, our candidate keys would include CustomerID, Name, and Email and we couldn't form a composite key anymore. Also, even if we don't consider Name and Email as candidate keys individually since logically, it wouldn't make sense to do so, if we try to form a composite key out of them instead, there will be no non-prime attribute left to check for partial dependency.

Orders Table
For the Orders table, the data suggest that only the OrderID qualifies as the candidate key. This means we don't have a composite key thus there will be no room for partial dependency.

Step 3: Third Normal Form (3NF)

Books Table
The Books table is already in the 3rd normal form and we'll give two different scenarios to prove this. Say we logically base it on the column names where it would make sense to only have the BookID as the prime attribute while the rest are non-prime attributes (Title, Author, Year). Title alone cannot determine the author because there could be a book with the same title but have a different author. So when we're given a book title, we can't uniquely identify a specific author so we can't say that Author is dependent on Title. The same goes with Title and Year or Author and Year or Author and Title or Year and Title or Year and Author. There is no transitive dependency.

Now, if we base it from the data, BookID, Title, and Year would be prime attributes. That leaves us with Author as the sole non-prime attribute and it can't depend another non-prime attribute because there is none.

Customers Table
For customers table, both the data and the logical meaning of the column names suggest that CustomerID and Email are candidate keys and they both can be primary keys in themselves. Because based from the data, john@example.com will always give us John Doe. In the same way, in real life, a specific email address can only belong to one person. This leaves us with Name being the only non-prime attribute so it cannot depend on another non-prime attribute making the table pass the 3NF criteria.

Orders Table
For the Orders table, both CustomerID and BookID are non-prime attributes. However none of them depend on each other because a CustomerID does not uniquely identify a BookID. A CustomerID could borrow multiple books with different IDs. In the same way, based from the data, a BookID can be borrowed by multiple customers so given a BookID, we cannot uniquely identify a customerID. For all these reasons the Orders Table is already in the 3rd Normal Form.

2. Consider the following table, which represents customer orders:
OrderID | CustomerName | Product | Quantity | OrderDate

First Normal Form (1NF)

The table is already in 1NF because values are already atomic. Based on the column names, we are expecting only a single value in each of these columns, not a list, or a collection of items. We also have a primary key (OrderID, Product) which can uniquely identify the row.

Second Normal Form (2NF)

However, the given table is not yet in 2NF and here's why. Both the OrderID and the Product form the primary key because we could have different orders from a customer. We can liken the orderID as the current checkoutID of Amazon for example. An order could contain multiple products with varying quantities but an order ID can only belong to one customer and it should only have one OrderDate because this is one of the factors that make them belong to one order - if they were checked-out together meaning on the same date. Having said that, since we are able to uniquely identify CustomerName and OrderDate through orderID and our primary key is a composite key consisting of OrderId and Product, CustomerName and OrderDate has a partial dependency to OrderID. Therefore we have to put them on a separate table hence our revised table that is now in 2NF:

Orders Table

OrderID | CustomerName | OrderDate

Order Details Table

OrderID | Product | Quantity

Third Normal Form (3NF)

Orders Table

Candidate Keys:
OrderID

Non-Prime Attributes
CustomerName
OrderDate

Order Details Table

Candidate keys:
(OrderID, Product)

Non-Prime Attributes
Quantity

As we can see from above, starting with Order Details table, there is only one non-prime attribute so a transitive dependency couldn't exist. For the Orders Table, CustomerName is not dependent on OrderDate because given an OrderDate, we could end up with multiple customers because possibly other customers also ordered other items on that date. In the same way, OrderDate is also not dependent on CustomerName because given a customer name, we could end up having different dates in the result set because possibly, a customer made another order the next day or any other date. Thus neither of these non-prime attributes depend on each other satisfying our 3rd Normal Form criteria.

3. Given the following unnormalized table structure for a university course enrollment system:
StudentID | StudentName | Major | Course1 | Course2 | Course3 | ... | CourseN

First Normal Form
This is a bad design because it attempts to list the various courses taken by a student through creating a column for each. This design contains repeating groups thus violating the 1NF rule. To address this, we could do two things.

First, we keep only one course column and if a student takes 2 or more courses these should be additional entries to the table. For example:

StudentID	**StudentName**	**Major**	**Course**
1	Umur	CS	DBMS
1	Umur	CS	EA

Another way to go about is create a separate table for StudentID and Course:

Student Table
StudentID | StudentName | Major

Student_Courses Table
StudentID | Course

Both of these will now satisfy our 1NF rule.

Second Normal Form
To convert these tables to 2NF. Starting with Student Table, we will assume thtat a student can take more than 1 major as this is the most logical in real life. People can graduate with a double major. So, we have to make our StudentID and Major our composite primary key. Since StudentName(our sole non-prime attribute) depends on StudentID, meaning given a StudentID we can uniquely identify a name, or in other words if we are given StudentID : 1, we will always get the result Umur, StudentName has a partial dependency to StudentID, so we have to extract both of these columns because they violate the 2NF rule. For the Student_Courses Table, both StudentID and the Course are part of the composite primary key so a partial dependency couldn't exist. Applying what was discussed above, the new table in 2NF now looks like:

Student Table
StudentID | StudentName

Student_Major Table
StudentID | Major

Student_Course
StudentID | Course

Third Normal Form
Looking at the tables above, for Student Table, there is only one non-prime attribute (StudentName) so a transitive dependency couldn't exist. For the Student_Major Table, both the StudentID and the Major are part of the composite primary key so a transitive dependency also couldn't exist. The same applies to StudentID and Course in the Student_Course Table because both of the columns form the primary key of the table.