Task A:

Given relations are:

Buyer(Buyer_id, Buyer_Name, Email, Address)
Seller(Seller_id, Seller_Name, Email, Address)
Authenticate(Auth_id, Password)
Product(P_id, Title, Description, Price, Seller_id)
Payment(Payment_id, P_id, B_id, Amount, Payment_time)
Bid(Bid_id, B_id, P_id)
Listings(Listing_id, Status, Start_date, End_date, Seller_id)

- The above relations are not in BCNF. To convert it into BCNF, we need to check all normal forms including 1NF, 2NF and 3NF.

1st Normal form (1NF):

To convert the given database schema into first normal form (1NF):

- we need to ensure that there are no repeating groups or arrays within the tables.
- Each attribute should contain only atomic values, and each table should have a primary key to uniquely identify each row.

Schema in 1NF:

Buyer(Buyer_id, Buyer_Name, Email, Address)
Seller(Seller_id, Seller_Name, Email, Address)
Authenticate(Auth_id, Password)
Product(P_id, Title, Description, Price, Seller_id)
Payment(Payment_id, P_id, B_id, Amount, Payment_time)
Bid(Bid_id, B_id, P_id)
Listings(Listing_id, Seller_id)

Functional dependencies in the relations:

- Functional dependencies describe the relationship between attributes in a table, where knowing the value of one or more attributes determines the value of another attribute.

The functional dependencies for each table in the given schema:

Table	Functional dependencies
Buyer	• Buyer_id -> Buyer_Name, Email, Address • Email -> Buyer_id
Seller	• Seller_id -> Seller_Name, Email, Address • Email -> Seller_id
Authenticate	• Auth_id -> Password
Product	• P_id -> Title, Description, Price, Seller_id • Seller_id -> P_id
Payment	• Payment_id -> Bid_id, B_id, Amount, Payment_time • Bid_id -> Payment_id
Bids	• Bid_id -> B_id, P_id • B_id -> Bid_id • P_id -> Bid_id
Listings	• Listing_id -> Seller_id, Status, Start_date, End_date • Seller_id -> Listing_id

2nd Normal form (2NF):

To convert the given schema into second normal form (2NF), we need to ensure that all non-key attributes in each table depend on the entire primary key, rather than just a part of it.

To achieve this, we need to identify any attributes that depend only on a subset of the primary key and move them to a separate table along with the subset of the key they depend on.

Here's the 2NF schema with their functional dependencies and partial dependencies if any:

Table 1: Buyer

- Buyer_id (PK)
- Email (unique)
- Buyer_Name
- Address

Functional Dependencies	Partial dependencies
• Buyer_id -> Email, Buyer_Name, Address • Email -> Buyer_id	None

Explanation: The table is already in 1NF. The primary key is Buyer_id and Email is unique. Buyer_Name and Address depend on the entire primary key, so there are no partial dependencies.

Table 2: Seller

- Seller_id (PK)
- Email (unique)
- Seller_Name
- Address

Functional Dependencies	Partial dependencies
• Seller_id -> Email, Seller_Name, Address • Email -> Seller_id	None

Explanation: The table is already in 1NF. The primary key is Seller_id and Email is unique. Seller_Name and Address depend on the entire primary key, so there are no partial dependencies.

Table 3: Authenticate

- Auth_id (PK)
- Password

Functional Dependencies	Partial dependencies
• Auth_id -> Password	None

Explanation: The table is already in 1NF. The primary key is Auth_id and there are no partial dependencies.

Table 4: Product

- P_id (PK)
- Title
- Description
- Price
- Seller_id (FK)

Functional Dependencies	Partial dependencies
• P_id -> Title, Description, Price, Seller_id • Seller_id -> P_id	None

Explanation: The table is already in 1NF. The primary key is P_id and Seller_id is a foreign key. Title, Description, and Price depend on the entire primary key, so there are no partial dependencies.

Table 5: Payment

- Payment_id (PK)
- Amount
- Payment_time
- Bid_id (FK)
- B_id (FK)

Functional Dependencies	Partial dependencies
• Payment_id -> Amount, Payment_time, Bid_id, B_id • Bid_id -> Payment_id • B_id -> Payment_id	None

Explanation: The table is already in 1NF. The primary key is Payment_id and Bid_id and B_id are foreign keys. Amount and Payment_time depend on the entire primary key, so there are no partial dependencies.

Table 6: Bids

- Bid_id (PK)
- B_id (FK)
- P_id (FK)

Functional Dependencies	Partial dependencies
• Bid_id -> B_id, P_id • B_id -> Bid_id	None

• P_id -> Bid_id	

Explanation: The table is already in 1NF. The primary key is Bid_id and B_id and P_id are foreign keys. There are no partial dependencies.

Table 7: Listings

- Listing_id (PK)
- Status
- Start_date
- End_date
- Seller_id (FK)

Functional Dependencies	Partial dependencies
• Listing_id -> Status, Start_date, End_date, Seller_id • Seller_id -> Listing_id	None

Explanation: The table is already in 1NF. The primary key is Listing_id and Seller_id is a foreign key. Status, Start_date, and End_date depend on the entire primary key, so there are no partial dependencies.

Note: In the 2NF schema, there are no partial dependencies, but there are still transitive dependencies, such as the dependency between Seller_id and P_id in

3rd Normal form (3NF):

To convert the relation into 3NF, we need to check for transitive dependencies. A transitive dependency occurs when a non-key attribute depends on another non-key attribute.

The schema is already in 3NF and does not contain any transitive dependencies. Each table has a primary key, and the foreign keys are used to establish relationships between the tables. The functional dependencies have been identified and normalized to eliminate redundancy and improve data consistency.

Boyce-Codd Normal form (BCNF):

To convert the schema to BCNF, we need to identify all functional dependencies and ensure that all non-trivial dependencies have determinants that are superkeys.

Functional dependencies:

- Buyer_id → Buyer_Name, Email, Address
- Seller_id → Seller_Name, Email, Address

- Auth_id → Password
- Seller_id → {P_id, Title, Description, Price, Listing_id, Status, Start_date, End_date}
- Buyer_id, P_id → Bid_id
- Bid_id → {B_id, P_id, Amount, Payment_time}
- Seller_id → Listing_id
- Seller_id, Listing_id → {P_id, Title, Description, Price, Status, Start_date, End_date}
- B_id → {Buyer_Name, Email, Address}

It is possible to decompose the schema into smaller tables while still preserving dependencies, but it would not necessarily be lossless.

For example, we could decompose the schema as follows:

Table 1: Buyer (Buyer_id, Buyer_Name, Email, Address)

Table 2: Seller (Seller_id, Seller_Name, Email, Address)

Table 3: Authenticate (Auth_id, Password, Buyer_id)

Table 4: Product (P_id, Title, Description, Price, Seller_id)

Table 5: Payment (Payment_id, Bid_id, B_id, Amount, Payment_time)

Table 6: Bids (Bid_id, B_id, P_id)

Table 7: Listings (Listing_id, Seller_id, Status, Start_date, End_date)

This decomposition preserves all dependencies, but it is not lossless because we have split the Authenticate table away from the Buyer table, which could result in some loss of information if we were to join these tables back together.

To achieve lossless decomposition, we could use an alternative decomposition that involves creating additional tables to represent certain relationships between entities. However, this would result in more tables and more complex relationships between them, which may not be desirable in practice.

The schema is in BCNF.