k230013
Zunaira Amjad
BAI-5A

DATABASE
A2

Q1.

A. Identify all primary Entities and their corresponding Attributes from the scenario. Underline the PrimaryKey (PK) for each entity.

Film

a. FilmID
b. Title
c. Budget
d. ReleaseDate
e. DirectorName

Actor

a. ActorID
b. Name
c. Phone

ProductionCompany

a. CompanyID
b. Name

B. Identify the three main Relationships (e.g., A relates to B). State the Cardinality (1:1, 1:M, M:N) for each relationship.

Film – Actor

a. Relationship: *Casts*
b. Cardinality: M:N (Many-to-Many)
c. Reason: A film can have many actors; an actor can be in many films.

Film – ProductionCompany

a. Relationship: *FundedBy*
b. Cardinality: M:N (Many-to-Many)
c. Reason: A film can be funded by many companies; a company can fund many films.

Film – Director

a. Not a separate entity here; director is an attribute of Film.
b. So, no separate relationship table for Directors in this model.

C. Rule 3 describes a Many-to-Many (M:N) relationship. Name the Associative Entity (Linking Table) needed to resolve it and list all the attributes it will contain (including the PK).

Associative Entity Name: Casting
Attributes:

a. FilmID (FK to Film)
b. ActorID (FK to Actor)
c. CharacterName
d. PK -> Composite: FilmID, ActorID

D. **Explain why DirectorName (Rule 4) should NOT be made into a separate entity in this simplified scenario.**
DirectorName is stored directly in the Film table because no additional director attributes are being tracked, there's no data reuse requirement. So, if the same director directs multiple films, we simply store the name repeatedly without a foreign key link—this simplifies the schema by avoiding an extra table and joining when only the director's name is needed per film, and the business rules do not require tracking directors independently of films.

Q2.

Entities and Their Attributes

AIRLINE
AirlineCode
AirlineName
Country

CITY
CityID
CityName
Country

FLIGHT
FlightNumber
AirlineCode
BusinessClassIndicator
SmokingAllowedIndicator
OriginCityID (FK to CITY)
DestinationCityID (FK to CITY)

FLIGHT_AVAILABILITY
FlightNumber

AirlineCode
DepartureDateTime
TotalBusinessSeats
BookedBusinessSeats
TotalEconomySeats
BookedEconomySeats

CUSTOMER
CustomerID
FirstName
LastName
Street
City
ProvinceState
PostalCode
Country

CUSTOMER_PHONE (Weak Entity)
CustomerID_ (FK to CUSTOMER)
PhoneNumber

CUSTOMER_FAX (Weak Entity)
CustomerID (FK to CUSTOMER)
FaxNumber

CUSTOMER_EMAIL (Weak Entity)
CustomerID (FK to CUSTOMER)
EmailAddress

CURRENCY
CurrencyCode
CurrencyName

EXCHANGE_RATE
CurrencyCode (FK to CUSTOMER)
Date
RateToUSD

BOOKING
BookingNumber

BookingCityID (FK to CITY)
BookingDate
FlightNumber (FK to FLIGHT)
AirlineCode (FK to FLIGHT)
DepartureDateTime (FK to FLIGHT_AVAILABILITY)
ArrivalDateTime
ClassIndicator
FlightPrice
OriginAirportTax
DestinationAirportTax
TotalPrice – Calculated as (FlightPrice + OriginAirportTax + DestinationAirportTax)
Status
ResponsibleCustomerID (FK to CUSTOMER)
AmountPaidSoFar
OutstandingBalance
TicketFirstName
TicketLastName

Relationships and Cardinality

AIRLINE operates FLIGHT (1:M)

- One airline operates many flights. A flight is operated by one airline.
- Represented by the AirlineCode FK in the FLIGHT table.

FLIGHT has availability for FLIGHT_AVAILABILITY (1:M)

- One flight (route) has many specific instances (availability) on different dates/times. A specific availability instance belongs to one flight.
- Represented by the composite FK (FlightNumber, AirlineCode) in FLIGHT_AVAILABILITY

CUSTOMER makes BOOKING (1:M)

- One customer can make many bookings. A booking is made by one responsible customer.
- Represented by the ResponsibleCustomerID FK in the BOOKING table.

FLIGHT_AVAILABILITY is booked by BOOKING (1:M)

- One specific flight instance can have many bookings. A booking is for one specific flight instance.

- Represented by the composite FK (FlightNumber, AirlineCode, DepartureDateTime) in BOOKING.

CITY is the origin/destination for FLIGHT (M:N)

- This is resolved via the OriginCityID and DestinationCityID attributes in the FLIGHT entity itself, creating two M:1 relationships.
- One city can be the origin for many flights and the destination for many flights.

CUSTOMER has PHONE/FAX/EMAIL (1:M for each)

- One customer can have zero or more phone numbers, fax numbers, and email addresses.
- Represented by the weak entities CUSTOMER_PHONE, CUSTOMER_FAX, and CUSTOMER_EMAIL, which have `CustomerID` as part of their PK and an FK to CUSTOMER

Q3.

a. 1. Insertion Anomaly

It is impossible to add a new school to the database until at least one teacher has been assigned a contract to work there. This forces the insertion of dummy or null data for teacher-related fields.

2. Deletion Anomaly

If the record for teacher "Lewis H" (NIN 2205) working at "Green School" (S114) is deleted, we lose the information that "Green School" is located in "Birmingham" if this was the only record for that school.

3. Update Anomaly

The school name "Park School" and city "Manchester" are repeated for every teacher assigned to that school. If the school's name changes, we must update multiple records. Inconsistent updates could lead to the same school having different names in different records.

b. Step 1: Identify Functional Dependencies (FDs)

1. schoolID → schoolName, schoolCity
    - A school ID determines the school's name and city.
2. NIN → tName
    - A National Insurance Number (NIN) determines the teacher's name.
3. contractNo, NIN, schoolID → hours
    - The hours a teacher works depend on a specific combination of contract, teacher, and school. This is the composite primary key for the original table.

Assumption: A teacher (NIN) can have the same contractNo but work at multiple schools, hence schoolID is part of the key to determine hours.

Step 2: Convert to 1NF (First Normal Form)

A table is in 1NF if it has no repeating groups and all attributes contain atomic values.

Our table is already in 1NF because:

- Each column contains atomic (indivisible) values.
- Each row is unique, identified by the composite key: (contractNo, NIN, schoolID).

So, the initial table is our starting 1NF relation:

Assignment (contractNo, NIN, schoolID, hours, tName, schoolName, schoolCity)

PK: (contractNo, NIN, schoolID)

Step 3: Convert to 2NF (Second Normal Form)

A table is in 2NF if it is in 1NF and has no partial dependency (i.e., no non-prime attribute depends on only part of the candidate key).

From our FDs:

- schoolID → schoolName, schoolCity → Here, schoolName and schoolCity depend only on schoolID (part of the key). This is a partial dependency.
- NIN → tName → Here, tName depends only on NIN (part of the key). This is also a partial dependency.
- contractNo, NIN, schoolID → hours → hours depends on the full key. This is fine.

We decompose to remove partial dependencies:

Relation 1: School

- schoolID (PK)
- schoolName
- schoolCity

Relation 2: Teacher

- NIN (PK)
- tName

Relation 3: Assignment (the fact of a teacher working hours under a contract at a school)

- contractNo
- NIN (FK to Teacher)
- schoolID (FK to School)
- hours
- PK: (contractNo, NIN, schoolID)

Step 4: Convert to 3NF (Third Normal Form)

A table is in 3NF if it is in 2NF and has no transitive dependencies (i.e., no non-prime attribute depends on another non-prime attribute).

Check our decomposed relations:

- School: schoolID → schoolName, schoolCity. No transitive dependencies. ✓
- Teacher: NIN → tName. No transitive dependencies. ✓
- Assignment: All non-key attributes depend on the key, the whole key, and nothing but the key. No transitive dependencies. ✓

Therefore, our relations are already in 3NF.

Final 3NF Schema

1. Teacher (NIN, tName)
    - PK: NIN
2. School (schoolID, schoolName, schoolCity)
    - PK: schoolID
3. Assignment (contractNo, NIN, schoolID, hours)
    - PK: (contractNo, NIN, schoolID)
    - FK: NIN references Teacher(NIN)
    - FK: schoolID references School(schoolID)

Q4.

A. Identify the Primary Key (PK) for the FILM_CASTING table based on the given Functional Dependencies.

FilmID, ActorID

B. The table is currently in 1NF. Which is the highest Normal Form (NF) it violates (2NF or 3NF)? Justify your answer by giving an example of a dependency violation.

The table is in 1NF (atomic values).
For 2NF:

- 2NF requires no partial dependency (i.e., no non-prime attribute depends on only part of the PK).

Here:

- FilmID → FilmTitle, DirectorName → FilmTitle and DirectorName depend only on FilmID (part of PK).
- ActorID → ActorName, AgentID → ActorName and AgentID depend only on ActorID (part of PK).

These are partial dependencies.

Violates 2NF because of partial dependencies.

C. Decompose the table to Second Normal Form (2NF). List the new tables created and state the specific dependencies that were removed.

Remove partial dependencies.

New tables:

1. FILM
    - FilmID (PK)
    - FilmTitle
    - DirectorName
    - FD: FilmID → FilmTitle, DirectorName
2. ACTOR
    - ActorID (PK)
    - ActorName
    - AgentID
    - FD: ActorID → ActorName, AgentID
3. CASTING
    - FilmID (FK to FILM)
    - ActorID (FK to ACTOR)
    - PK: (FilmID, ActorID)
    - FD: FilmID, ActorID → (no other attributes here)

Dependencies removed:

- FilmID → FilmTitle, DirectorName moved to FILM table.
- ActorID → ActorName, AgentID moved to the ACTOR table.

D. Decompose the 2NF tables further to Third Normal Form (3NF). List all final tables in their 3NF form with their PK and FK

Checking 3NF: No transitive dependencies (non-key attribute depends on another non-key attribute).

- FILM is in 3NF (all attributes depend on key FilmID).
- ACTOR has ActorID → AgentID and AgentID → AgentPhone (transitive dependency).
    - So AgentPhone depends on AgentID, not directly on ActorID.
    - Violates 3NF.

Decompose ACTOR:

1. ACTOR
    - ActorID (PK)
    - ActorName
    - AgentID (FK to AGENT)
    - FD: ActorID → ActorName, AgentID
2. AGENT
3. AgentID (PK)
4. AgentPhone
5. FD: AgentID → AgentPhone

CASTING remains the same.

Final 3NF tables:

- FILM (FilmID, FilmTitle, DirectorName)
- ACTOR (ActorID, ActorName, AgentID)
- AGENT (AgentID, AgentPhone)
- CASTING (FilmID, ActorID)
    - FK: FilmID references FILM
    - FK: ActorID references ACTOR

E. In the original FILM_CASTING table, AgentPhone is stored repeatedly for every film an actor is in. Which type of data redundancy anomaly (Insert, Update, or Delete) would this cause if the agent's phone number changes?

Update Anomaly

Q5.

1. Identify the functional dependencies represented by the attributes shown in the form above.State any assumptions that you make about the data and the attributes.

Assumptions:

- Each Course ID is assigned to one course name, fixed number of credits, and one instructor.
- An Instructor Name can teach multiple courses.
- A Student has one first name, last name, and address.
- A Student can enroll in many courses, and a course can have many students.
- Enrollment Date depends on both Student ID and Course ID (when the student enrolled in that course).

Functional Dependencies:

1. StudentID → FirstName, LastName, Address
2. CourseID → CourseName, Credits, InstructorName
3. InstructorName → (No, because one instructor can teach multiple courses, so InstructorName does not determine CourseID.)
4. StudentID, CourseID → EnrollmentDate
5. CourseID → InstructorName (but not the reverse)

2. Describe and illustrate the process of normalizing the attributes shown in the form above to produce a set of well-designed 3NF (Third Normal Form) relations.

Step 1: Unnormalized form

Initial table:
Enrollment (StudentID, FirstName, LastName, Address, CourseID, CourseName, Credits, InstructorName, EnrollmentDate)
PK: (StudentID, CourseID)

Step 2: 1NF

Already in 1NF (atomic values).

Step 3: 2NF – Remove partial dependencies

- StudentID → FirstName, LastName, Address (depends only on part of PK)
- CourseID → CourseName, Credits, InstructorName (depends only on part of PK)
- (StudentID, CourseID) → EnrollmentDate (depends on full key)

Decompose:

Student (StudentID, FirstName, LastName, Address)
Course (CourseID, CourseName, Credits, InstructorName)
Enrollment (StudentID, CourseID, EnrollmentDate)

Step 4: 3NF – Remove transitive dependencies

In Course table:
CourseID → InstructorName, and InstructorName → ?
We assumed InstructorName does not determine CourseID, but does InstructorName determine Credits or CourseName? No.
But: If InstructorName is the only instructor for a course, no transitive dependency in the Course table.
But if we store Instructor details separately, we'd need InstructorID. Here, InstructorName is directly in Course, so no transitive dependency unless we consider instructor's own attributes (office, phone) — none given, so Course is in 3NF.

check: Is InstructorName → Credits? No.
Is CourseID → InstructorName? Yes but fine

Actually, InstructorName could teach multiple courses, so no FD from InstructorName to CourseID or Credits.
So the course table is in 3NF.

But if we want to avoid update anomalies if the instructor changes name, better to have Instructor as a separate entity. But not required by 3NF unless there's a transitive dependency.

Given the data, all tables are in 3NF already after 2NF step because:

- Student: all attributes depend on StudentID.
- Course: all attributes depend on CourseID.
- Enrollment: all attributes depend on (StudentID, CourseID).

No transitive dependency exists in these.

Final 3NF relations:

1. Student (StudentID, FirstName, LastName, Address)
2. Course (CourseID, CourseName, Credits, InstructorName)
3. Enrollment (StudentID, CourseID, EnrollmentDate)

3. Identify the primary, alternate, and foreign keys in your 3NF relations.

Primary Keys:

- Student: StudentID
- Course: CourseID
- Enrollment: (StudentID, CourseID)

Alternate Keys:
None specified, but CourseName could be unique (assume it is), so CourseName is an alternate key in Course table.

Foreign Keys:

- Enrollment.StudentID → Student.StudentID
- Enrollment.CourseID → Course.CourseID