Lab 03- From Conceptual to Logical: Transforming our Models

Vishakha Maruti Sonmore (CSULB ID: 032188141)

College of Business, California State University, Long Beach

Course Number: I S 680 Sec02 11140 Database Management Systems

Instructor: Dr. Amini, Mostafa

Due date: 29- Feb-2024

Week 4 – Lab #3 – From Conceptual to Logical: Transforming our Models.

Table of Contents

Transforming Conceptual EERDs into Logical Models

Welcome to your third lab in IS680! Previously on IS680 we saw how to develop conceptual models through the Entity Relationship Diagrams and Enhanced Entity Relationship Diagrams. These conceptual models were meant to inform potential users and developers about the overall needs and requirements that our databases will need to satisfy. Nevertheless, they are not intended to be "implemented" As-Is.

To create an actual database, we must first transform our conceptual model into a form that can preserve the integrity of the data and ensure its consistency. This is the job of the Logical model. In this lab, we will go through the **process of transforming the components of the conceptual model into a logical model** that we can implement.

To finalize this lesson, we will also discuss the process of normalization and its multiple steps. Normalization consists of a series of checks and transformations that ensures our logical models are consistent and maintain the integrity of the data. It is important to note that if we performed the transformations noted above correctly, our logical model should already be normalized.

Lab Goals

- **SLO1**- To describe the differences between conceptual and logical models.
- **SLO2**- To transform conceptual models into logical models (i.e. transform EERDs into relations).
- **SLO3**- To create effective logical models with tables and relational integrity constraints.
- **SLO4**- To use the normalization process to correct poor logical models into well-structured relations.

What You Will Need to Begin

- Visit draw.io (https://www.draw.io) to gain access to a modeling tool either through its web version or by downloading a copy into your own computer.
- Please complete the modeling exercises in draw.io and copy your models (with your name visible) into this document. Screenshots are an acceptable choice for copying the models.
- Once completed, submit the exercise (this document) into a corresponding DropBox folder within BeachBoard.

Part 2 – Transforming Models.

To strengthen our modeling skills, we will now explore multiple examples of transformations from conceptual models into logical models. These transformations include:

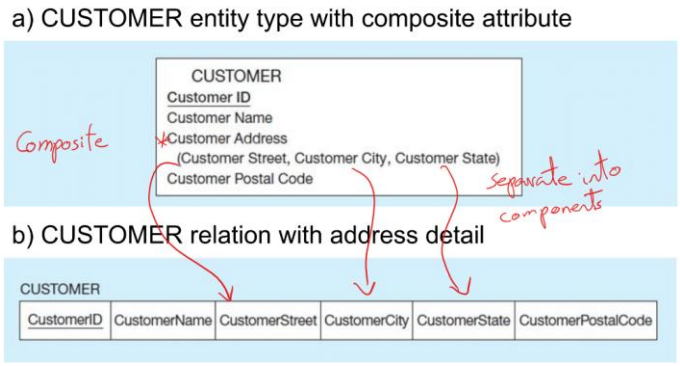
- Simple attributes
- Composite attributes
- Multivalued attributes
- Weak entities
- Binary relationships: One-to-Many, Many-to-Many, One-to-One
- Associative entities
- Unary relationships: One-to-Many, Many-to-Many, One-to-One
- Supertype/Subtype relationships

Transforming Attributes:

When transforming an EERD into a logical model we are moving from a model based on entities, attributes, and relationships into a model based on tables (known as relations which is different from relationships), attributes and Primary Key/Foreign Key-based relationships. In these transformations every entity will become a table, and attributes may remain the same or change to ensure the consistency of the data. The possible changes are:

Simple attributes: Simple attributes (i.e. those that represent a single value per cell) are kept as regular attributes in the logical model.

Composite attributes : Composite attributes (i.e. those with multiple pieces of data within the same attribute such as an address) must be separated into its components with 1 attribute per component. For example:



In this example we can clearly see that 1 composite attribute called “customer address” was separated in its components with each component turning into its own attribute. Specifically, it became 3 different attributes:

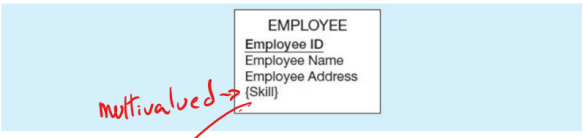
1) Customer street, 2) Customer city, and 3) Customer state.

By doing this, we now eliminated the composite attribute and are left with only simple attributes.

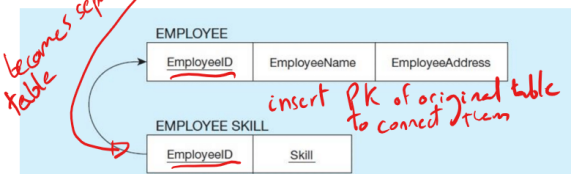
1. Multivalued attributes

Multivalued attributes can have any number of components which can in turn vary significantly among each other. This leads to significant repetition of data, difficulty keeping track of changes related to any of its possible values, among other issues. Therefore, it is necessary to simplify them to prevent problems in the first place. We do this by extracting any multivalued attribute and turning it into its own table/relation. For example:

a) EMPLOYEE entity type with multivalued attribute



b) EMPLOYEE and EMPLOYEE SKILL relations



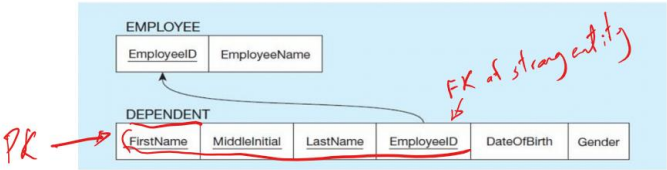
Apart from the attributes, it is also important to recognize how to transform other types of entities such as weak entities and associative entities (as well as associative attributes). Let’s review these:

2. Weak entities

Weak entities (entities representing something whose record depends on a different entity (a strong entity). Weak entities become tables/relations themselves with a foreign key to reference the strong entity they relate to. The primary key them becomes the combination of the identifier of the weak entity and the primary key of the strong entity it relates to. For example:

a) Weak entity DEPENDENT

b) Relations resulting from weak entity



1. **Associative attributes and associative entities**

The representation of associative attributes can vary depending on the type of the relationship.

If the relationship is 1:1 (one-to-one), the attributes in the associative attribute become attributes in one of the entities involved in the relationship. If one side is mandatory and the other is optional, they will always go into the optional side.

If the relationship is 1:N (one-to-many), the attributes will go into the many side of the relationship (allowing for more detail to be stored).

If the relationship is M:N (many-to-many) the associative attribute will become its own table/relation capable of storing information about the matches between the 2 related entities in the conceptual model. The primary key will them become the composite of the primary keys (now foreign keys from the perspective of the new table).

Associative entities (entities recording the matches between 2 other entities) follow a similar pattern and become their own table/relation containing the attributes of the relationship. They may have primary keys composed of the combination of the primary keys (registered as foreign key in this new table) of the 2 entities being related. Alternatively, they can also have their own unique primary key separate from that matching. It is typically best to give a unique identifier to the table, but the combination of the primary keys of the 2 related entities in the conceptual model can be used specially if it results in a unique value or repetition of the same combination is not possible for a given set.

Expanding on these transformations, it is important for us to now consider how to transform relationships between entities (unary, binary, ternary, or n-ary).

Binary relationships can be summarized as follows:

- **One-to-Many** – The Primary on the one side becomes a foreign key on the many side.
- **Many-to-Many** – Create a new table/relation that will store the matches between the 2 original entities. The primary keys of each entity would be recorded in this new table/relation as foreign keys and combined they will be used as the composite primary key of the table.
- **One-to-One** – The primary key on the mandatory side becomes a foreign key on the optional side of the relationship.

Unary relationships can be summarized as follows:

Unary relationships can be seen as One-to-Many or Many-to-Many. If it is:

- **One-to-Many** – We store the primary key of the table one again as a separate row with a title relevant to the type of relationship that exists. For example, if we have a table with employee data, the new column with the identifier of the employee could be called "Manager" to signal that it is representing who the Manager is.
- **Many-to-Many** – If we have a many-to-many (M:N) Unary relationship, we simple follow the guidelines of M:N relationships stated above. That means that we create a separate table for the matches of the 2 entities in the conceptual model, and store the primary keys of each into it. The combination of those primary keys (now called foreign keys since they are stored in the new table) will them become the primary key of the new table.

Ternary & n-ary relationships can be summarized as follows:

Ternary (and higher) relationships will always result in the creation of an associative table that stores the information about the matches of each original entity (which will be referred to as table once they are stored in the logical model). This new associative relation will contain foreign keys referring to the primary keys of each related table and will have a relationship to each one of those related tables.

Exercise 1: Transforming Attributes

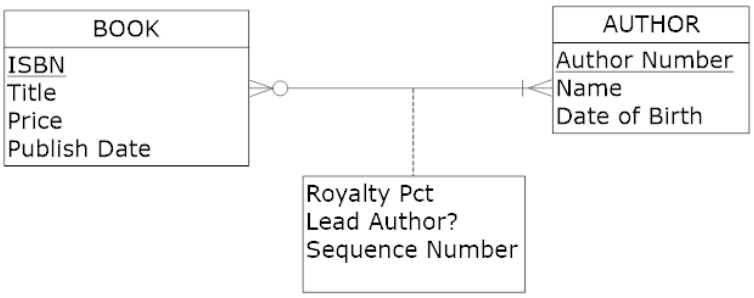
For exercise 1, please take the following models (based on those from lab#1) and transform them using the norms above to create your first logical model. The text the models is based on is written for your reference, but you should be able to do the transformation depending solely on the diagrams themselves.

For each problem, please provide an explanation to how you approach this transformation (why did you transformed it the way you did), and a screenshot of the logical you created.

Q1. Transform the solution for problem 3 into a logical model

NOTE: These problems are the same from lab 1 as found on the PPT "Practicing Data Modeling_Answers".

In the context specified above in problem 1 and problem 2, better information is needed regarding the relationship between a book and its authors. Specifically, it is important to record the percentage of the royalties that belong to a specific author, whether or not a specific author is a lead author of the book, and each author's position in the sequence of the book's authors.



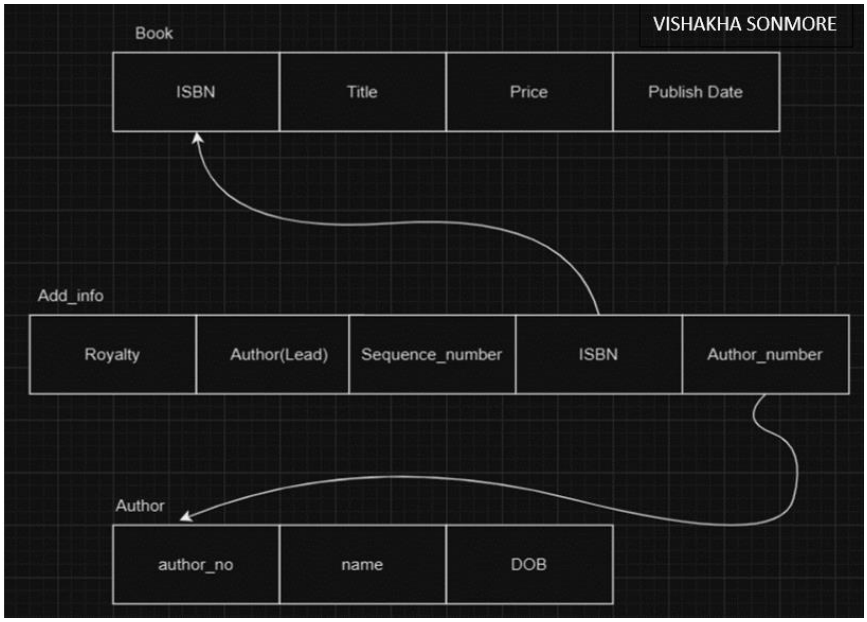
Problem 3: Solution

Logical Model:

- Book - Attributes: ISBN (Primary Key), Title, Price, Publication Date
- Author - Attributes: Author_Number (Primary Key), Name, Date of Birth
- Book_Author Table (Associative Entity):-
- Attributes: Book_ISBN (Foreign Key), Author_Number (Foreign Key), Royalty_Pct, Lead_Author (Boolean), Sequence_Number

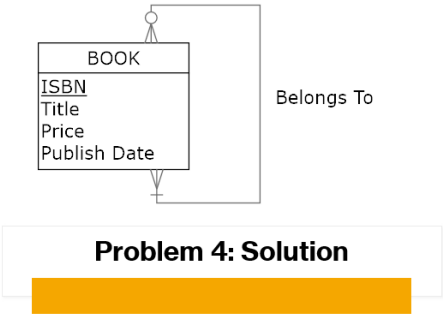
Explanation:

- Book Table: Maintained the Book entity with ISBN as the primary key. Added the Price attribute to represent the cost of the book.
- Author Table: Kept the Author entity with Author_Number as the primary key. Included attributes such as Name and Date of Birth to describe the author.
- Book_Author Table (Associative Entity): Introduced an associative entity, Book_Author, to represent the relationship between books and authors.
- Added attributes like Royalty_Pct to store the percentage of royalties, Lead_Author as a boolean to indicate if the author is the lead, and Sequence_Number to specify the author's position in the sequence.
- Ensured foreign key relationships between Book_Author and both Book (using Book_ISBN) and Author (using Author_Number) tables to maintain referential integrity.



Q2. Transform the solution for problem 4 into a logical model

D. A book (see (a) above) can be part of a series, which is also identified as a book and has its own I S B N number. One book can belong to several sets and a set consists of at least one but potentially many books.



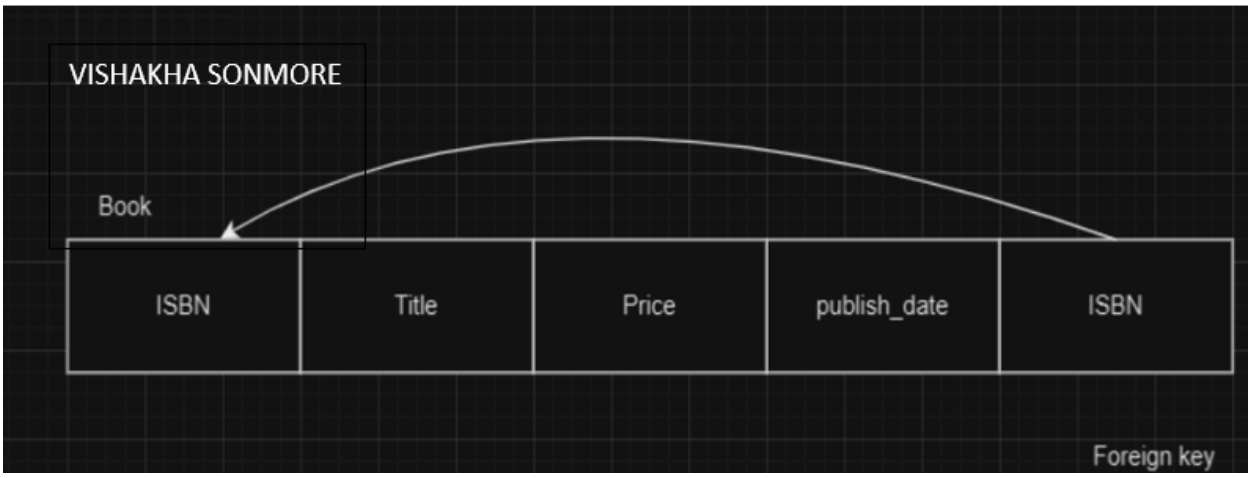
Logical Model:

- Book Table: Attributes: ISBN (Primary Key), Title, Price, Publication Date
- Book_Book Table (Associative Entity for Many-to-Many Self-Reference): Attributes: Book1_ISBN (Foreign Key), Book2_ISBN (Foreign Key), Relationship_Attribute1, Relationship_Attribute2

Explanation:

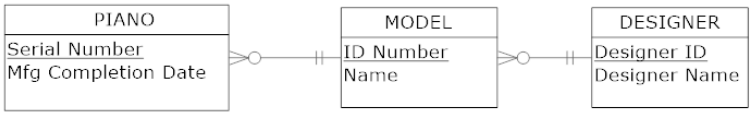
- Book Table: Maintained the Book entity with ISBN as the primary key. Includes attributes like Title, Price, and Publication Date.

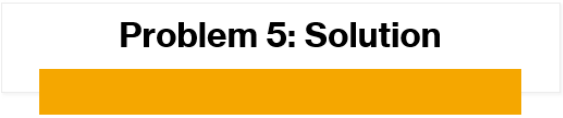
Created an associative entity, Book, to represent the many-to-many self-referencing relationship within the Book table. This model allows a book to be related to other books within the same table, and the optional attributes capture specific details about the relationship.



Q3. Transform the solution for problem 5 into a logical model

Ebony and Ivory, a piano manufacturer, wants to keep track of all the pianos it makes individually. Each piano has an identifying serial number and a manufacturing completion date. Each instrument represents exactly one piano model, all of which have an identification number and a name. In addition, the company wants to maintain information about the designer of the model. Over time, the company often manufactures thousands of pianos of a certain model, and the model design is specified before any single piano exists.



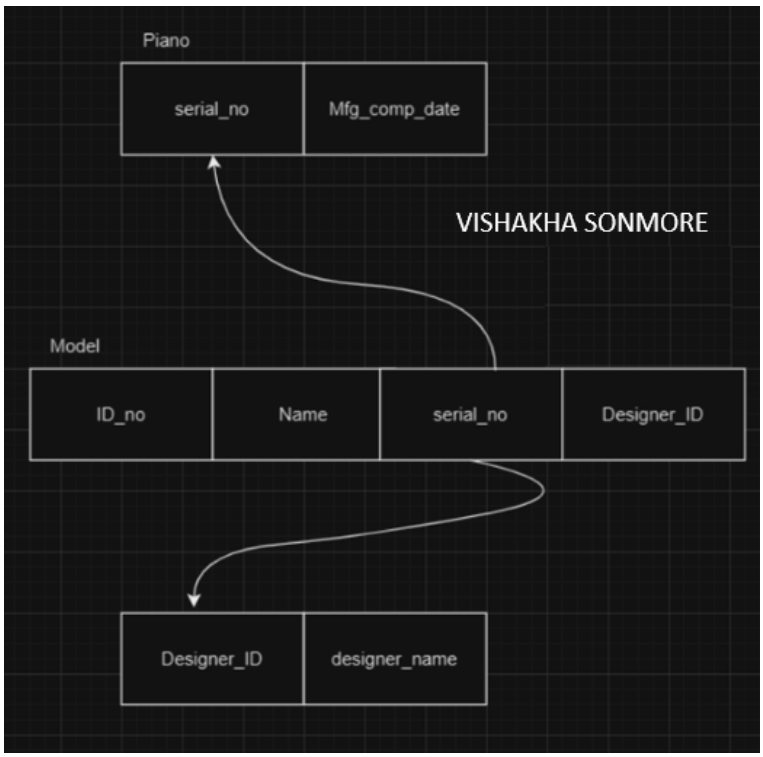


Logical Model:

- **Piano Table:**
  - Attributes: Serial_Number (Primary Key), Manufacturing_Completion_Date, Model_ID (Foreign Key), Designer_ID (Foreign Key)
- **Piano_Model Table:**
  - Attributes: Model_ID (Primary Key), Model_Name, Identification_Number
- **Designer Table:**
  - Attributes: Designer_ID (Primary Key), Designer_Name

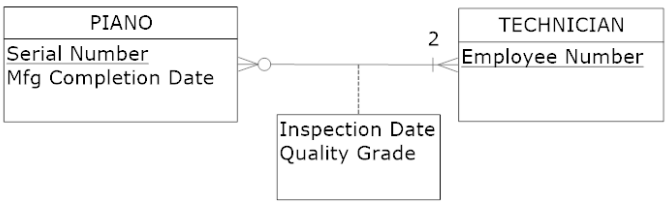
Explanation:

- **Piano Table:**
  - Represents each individual piano manufactured by Ebony and Ivory.
  - Includes attributes such as Serial_Number and Manufacturing_Completion_Date.
  - Foreign keys Model_ID and Designer_ID establish relationships with Piano_Model and Designer tables respectively.
- **Piano_Model Table:**
  - Represents the piano models manufactured by Ebony and Ivory.
  - Attributes include Model_ID, Model_Name, and Identification_Number.
- **Designer Table:**
  - Contains information about the designers of piano models.
  - Attributes include Designer_ID and Designer_Name.
- This model allows Ebony and Ivory to track individual pianos, associate them with specific models, and maintain information about the designers.



Q3. Transform the solution for problem 6 into a logical model

Ebony and Ivory (see (e) above) employs piano technicians who are responsible for inspecting the instruments before they are shipped to the customers. Each piano is inspected by at least two technicians (identified by their employee number). For each separate inspection, the company needs to record its date and a quality evaluation grade.

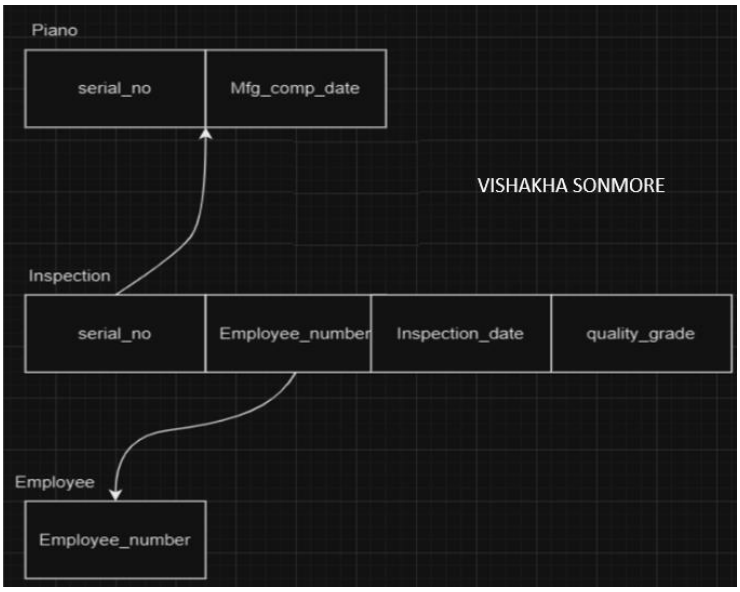


Problem 6: Solution

Explanation:

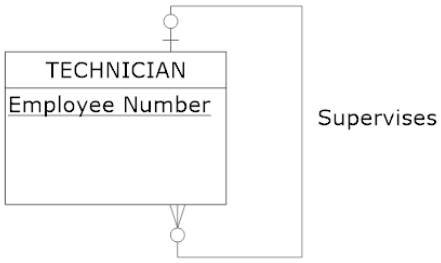
- Inspection Table:
  - Represents each inspection conducted by Ebony and Ivory.
  - Includes attributes such as Inspection_ID, Piano_Serial_Number, Technician1_ID, Technician2_ID, Inspection_Date, and Quality_Evaluation_Grade.
  - Foreign keys Technician1_ID and Technician2_ID establish relationships with the Technician table.

- Technician Table:
  - Represents the piano technicians employed by Ebony and Ivory.
  - Attributes include Technician_ID and Technician_Name.
- This model allows Ebony and Ivory to track inspections, associate them with specific technicians, and record details such as inspection date and quality evaluation grade.



Q3. Transform the solution for problem 7 into a logical model

The piano technicians (see (f) above) have a hierarchy of reporting relationships: some of them have supervisory responsibilities in addition to their inspection role and have multiple other technicians report to them. The supervisors themselves report to the chief technician of the company.



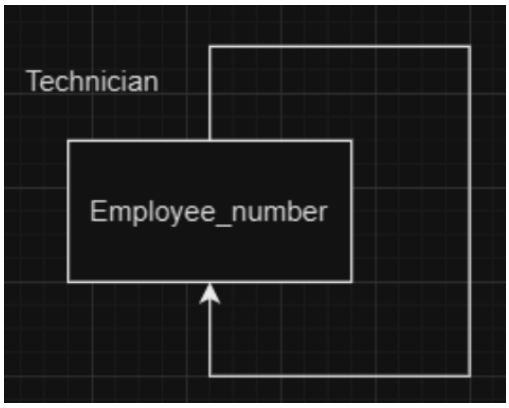
Problem 7:

Logical Model:

- **Technician Table:**
  - Attributes: Employer_Number

Explanation:

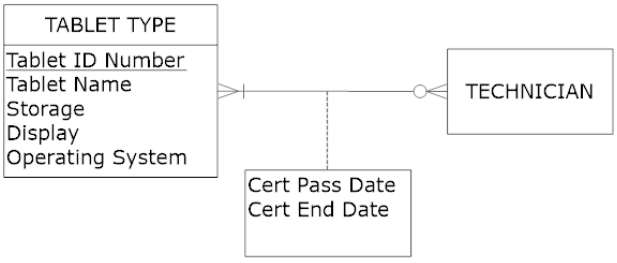
- **Technician Table:**
  - Represents the piano technicians employed by Ebony and Ivory.
  - Attributes include ID.
- This model captures the hierarchy where some technicians have supervisory responsibilities, and supervisors report to the chief technician.



Q3. Transform the solution for problem 10 into a logical model

Each of the tablet computer types (see (h) above) has a specific operating system. Each technician the company employs is certified to assemble a specific tablet type – operating system combination. The validity of a certification starts on the day the employee passes a certification examination for the

combination, and the certification is valid for a specific period of time that varies depending on tablet type – operating system combination.



Problem 10:

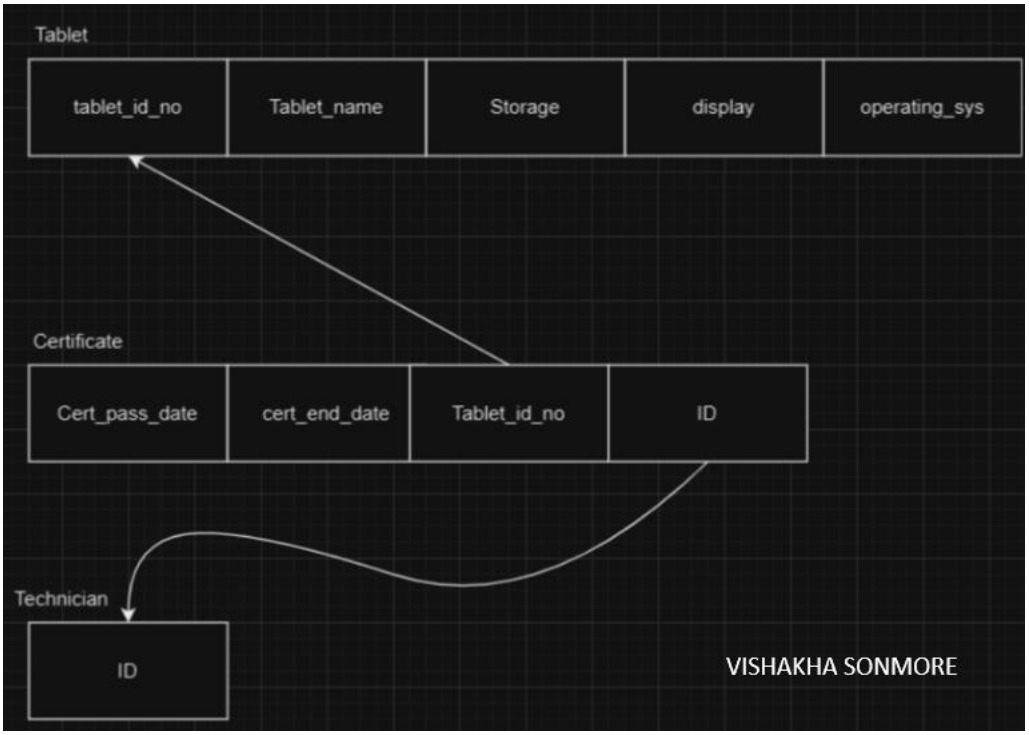
Logical Model:

- **Tablet_Type Table:**
  - Attributes: Type_ID (Primary Key), Type_Name, Storage, Display, Operating_System
- **Technician Table:**
  - Attributes: Technician_ID (Primary Key), Technician_Name
- **Certification Table:** Cert Pass date, Cert end date

Explanation:

- Tablet_Type Table:
  - Represents different types of tablet computers.
  - Includes attributes Type_ID, Type_Name, Storage, Display, and Operating_System.
- Technician Table:
  - Represents technicians employed by the company.
  - Includes attributes Technician_ID and Technician_Name.
- Certification Table:
  - Represents certifications obtained by technicians for assembling specific tablet types.
  - Includes foreign keys Technician_ID and Type_ID to link to Technician and Tablet_Type tables respectively.
  - Includes Certification_Pass_Date to record the date when the certification was obtained.
  - Includes Certification_End_Date to record the expiration date of the certification.

This model enables the company to track technician certifications for assembling specific tablet types, including details about the certification pass date and the expiration date of the certification.



Part 3: Practicing Normalization

Exercise 1: Transforming Attributes

The manager of a company dinner club would like to have an information system that assists him to plan the meals and to keep track of who attends the dinners, and so on.

Because the manager is not an IS expert, the following table is used to store the information. As a member can attend many dinners and a member will not attend more than 1 dinner on the same date, **the primary key of the following table is Member ID, Dinner ID, & FoodCode.** Dinners can have many courses, from one-course dinner to as many courses as the chef desired.

Member = customer

Dinner = meal

Venue = location

Food = specific food item

PK = MEMBERID + DINNERID + FOODCODE

TEST: Does each attribute depend on both components of the Primary Key, or just a portition?

MemberID	MemberName	Member Address	**DinnerID**	DinnerDate	VenueCode	VenueDescription	Food Code	FoodDescription
214	Peter Wong	325 Meadow Park	D0001	02/02/2020	L01	Grand_Ball_Room	EN3	Stu ed crab
							DEB	Beef
235	Mary Lee	123 Bellfower	D0002	02/02/2020	L02	Café	EN5	Chocolate Mousse
							DEB	Beef
250	John Doe	9923 Orange	D0003	03/03/20	L01	Grand_Ball_Room	SO1	Marinated Steak
							EN5	Chocolate Mousse
							DE2	Apple Pie
235	Mary Lee	123 Bellfower	D0004	03/03/2020	L02	Café	S01	Pumpkin Soup
							SA2	Marinated Steak
							DE2	Apple Pie
300	Paul Lee	123 Bellfower	D0004	03/03/2020	L03	Petit_Ball_Room	SA2	Marinated Steak

MemberID	MemberName	Member Address	DinnerID	DinnerDate	VenueCode	VenueDescription	Food Code	FoodDescription
214	Peter Wong	325 Meadow Park	D0001	02/02/2020	L01	Grand_Ball_Room	EN3	Stu ed crab
214	Peter Wong	325 Meadow Park	D0001	02/02/2020	L01	Grand_Ball_Room	DEB	Beef
235	Mary Lee	123 Bellfower	D0002	02/02/2020	L02	Café	EN5	Chocolate Mousse
235	Mary Lee	123 Bellfower	D0002	02/02/2020	L02	Cafe	DEB	Beef
250	John Doe	9923 Orange	D0003	03/03/20	L01	Grand_Ball_Room	SO1	Marinated Steak
250	John Doe	9923 Orange	D0003	03/03/20	L01	Grand_Ball_Room	EN5	Chocolate Mousse
250	John Doe	9923 Orange	D0003	03/03/20	L01	Grand_Ball_Room	DE2	Apple Pie
235	Mary Lee	123 Bellfower	D0004	03/03/2020	L02	Café	S01	Pumpkin Soup
235	Mary Lee	123 Bellfower	D0004	03/03/2020	L02	Café	SA2	Marinated Steak
235	Mary Lee	123 Bellfower	D0004	03/03/2020	L02	Café	DE2	Apple Pie

300	Paul Lee	123 Bellfower	D0004	03/03/2020	L02	Petit_Ball_Room	SA2	Marinated Steak

1. Is the above table considered a relation? Why or why not? Is it in any normal form? If so, which one?

The table presented does not meet the criteria of a relation as per the relational database model. It lacks the organization into distinct rows and columns, and it doesn't establish clear relationships between entities. The structure of the table suggests it is denormalized, featuring repeating data groups within the same row, resembling a flat file structure rather than a relational database table.

Regarding normalization, the provided table does not adhere to any normal form standards. Normalization standards are employed in relational databases to reduce redundancy and dependency anomalies. The table's structure violates the principles of the first normal form (1NF) due to the presence of repeating groups and multiple values within a single cell.

To comply with the rules of 1NF, the table requires restructuring into rows and columns, ensuring each attribute contains atomic values without repeating groups. This restructuring lays the groundwork for further normalization into higher normal forms such as 2NF and 3NF, aimed at resolving redundancy and dependency issues.

2. Transform the table above into first normal form 1NF. (To do this, check if there are multivalued attributes and transform the table to get rid of them)

To transform the table into the first normal form (1NF), we need to eliminate multivalued attributes and ensure that each cell in the table contains atomic values. Based on the provided table, it seems like we have repeating groups of data that need to be separated. Let's break down the table into atomic values:

Original Table (Before Transformation):

MemberID	MemberName	MemberAddress	DinnerID	DinnerDate	VenueCode	VenueDescription	FoodCode	FoodDescription
214	Peter Wong	325 Meadow Park	D0001	2/2/2020	L01	Grand_Ball_Room	EN3	Stuffed crab
							DEB	Beef
235	Mary Lee	123 Bellflower	D0002	2/2/2020	L02	Café	EN5	Chocolate Mousse
							DEB	Beef
250	John Doe	9923 Orange	D0003	3/3/2020	L01	Grand_Ball_Room	SO1	Marinated Steak
							EN5	Chocolate Mousse
							DE2	Apple Pie
235	Mary Lee	123 Bellflower	D0004	3/3/2020	L02	Café	S01	Pumpkin Soup
							SA2	Marinated Steak
							DE2	Apple Pie
300	Paul Lee	123 Bellflower	D0004	3/3/2020	L03	Petit_Ball_Room	SA2	Marinated Steak

Transformed Table (1NF):

MemberID	MemberName	MemberAddress	DinnerID	DinnerDate	VenueCode	VenueDescription	FoodCode	FoodDescription
214	Peter Wong	325 Meadow Park	D0001	2/2/2020	L01	Grand_Ball_Room	EN3	Stuffed crab
214	Peter Wong	325 Meadow Park	D0001	2/2/2020	L01	Grand_Ball_Room	DEB	Beef
235	Mary Lee	123 Bellflower	D0002	2/2/2020	L02	Café	EN5	Chocolate Mousse
235	Mary Lee	123 Bellflower	D0002	2/2/2020	L02	Café	DEB	Beef
250	John Doe	9923 Orange	D0003	3/3/2020	L01	Grand_Ball_Room	SO1	Marinated Steak
250	John Doe	9923 Orange	D0003	3/3/2020	L01	Grand_Ball_Room	EN5	Chocolate Mousse
250	John Doe	9923 Orange	D0003	3/3/2020	L01	Grand_Ball_Room	DE2	Apple Pie
235	Mary Lee	123 Bellflower	D0004	3/3/2020	L02	Café	S01	Pumpkin Soup
235	Mary Lee	123 Bellflower	D0004	3/3/2020	L02	Café	SA2	Marinated Steak
235	Mary Lee	123 Bellflower	D0004	3/3/2020	L02	Café	DE2	Apple Pie
300	Paul Lee	123 Bellflower	D0004	3/3/2020	L03	Petit_Ball_Room	SA2	Marinated Steak

In the transformed table, each row represents a unique combination of MemberID, MemberName, MemberAddress, DinnerID, DinnerDate, VenueCode, VenueDescription, FoodCode, and FoodDescription. We have eliminated the multivalued attributes by duplicating the necessary rows to ensure atomic values in each cell. This table now conforms to the first normal form (1NF).

3. **Identify the dependencies and which type they are (full dependencies, partial dependencies, transitive dependencies.**

To discern the dependencies within the provided table and categorize their types, an analysis of the functional dependencies among the attributes is required. The following dependencies have been identified:

- Full Dependencies:
    - MemberID, MemberName, MemberAddress → {DinnerID, DinnerDate, VenueCode, VenueDescription, FoodCode, FoodDescription}
    - DinnerID, DinnerDate, VenueCode → {VenueDescription}
    - FoodCode → {FoodDescription}
- Partial Dependencies:
    - No instances of partial dependencies were observed in this table.
- Transitive Dependencies:
    - No instances of transitive dependencies were observed in this table.

The table appears relatively straightforward, devoid of partial or transitive dependencies. Each attribute in the table exhibits full dependency on the entire primary key, and no non-prime attributes rely on a portion of the primary key. The functional dependencies identified above delineate the functional relationships among attributes within the table. These dependencies serve as foundational insights for subsequent normalization into higher normal forms.

4. **Transform the table above into second normal form 2NF. (To do these separate partial dependencies into separate tables).**

Let's consider the identified full dependencies:

1. MemberID, MemberName, MemberAddress → {DinnerID, DinnerDate, VenueCode, VenueDescription, FoodCode, FoodDescription}
2. DinnerID, DinnerDate, VenueCode → {VenueDescription}
3. FoodCode → {FoodDescription}

To achieve 2NF, we'll create separate tables for each identified dependency:

Table 1: Members

- MemberID (Primary Key)
- MemberName
- MemberAddress

Table 2: Dinners

- DinnerID (Primary Key)
- DinnerDate
- VenueCode (Foreign Key referencing Table 3)

Table 3: Venues

- VenueCode (Primary Key)
- VenueDescription

Table 4: Foods

- FoodCode (Primary Key)
- FoodDescription

By organizing the data into these separate tables, we address the partial dependencies and ensure that each table represents a single, logically related entity. This restructuring facilitates better data management and conforms to the principles of 2NF.

5. Transform the table above into third normal form 3NF. (To do this, remove the transitive dependencies by creating separate tables and relate them with the common attribute)

To transform the tables into the third normal form (3NF) and eliminate transitive dependencies, we need to identify and separate the related attributes into distinct tables.

Let's consider the existing tables:

Table 1: Members

- MemberID (Primary Key)
- MemberName
- MemberAddress

Table 2: Dinners

- DinnerID (Primary Key)
- DinnerDate
- VenueCode (Foreign Key referencing Table 3)

Table 3: Venues

- VenueCode (Primary Key)
- VenueDescription

Table 4: Foods

- FoodCode (Primary Key)

- FoodDescription

Now, let's identify transitive dependencies and create new tables:

Table 5: MemberDinners

- MemberDinnerID (Primary Key)
- MemberID (Foreign Key referencing Table 1)
- DinnerID (Foreign Key referencing Table 2)

By introducing the MemberDinners table, we eliminate the transitive dependency between Members and Dinners.

Here's the revised structure:

Table 1: Members

- MemberID (Primary Key)
- MemberName
- MemberAddress

Table 2: Dinners

- DinnerID (Primary Key)
- DinnerDate

Table 3: Venues

- VenueCode (Primary Key)
- VenueDescription

Table 4: Foods

- FoodCode (Primary Key)
- FoodDescription

Table 5: MemberDinners

- MemberDinnerID (Primary Key)
- MemberID (Foreign Key referencing Table 1)
- DinnerID (Foreign Key referencing Table 2)

This structure ensures that each table represents a single, logically related entity, and there are no transitive dependencies among the attributes.

Exercise 2: Normalizing Grade Reports

Figure 4-4 shows a relation called GRADE REPORT for a university. Your assignment is as follows:

Table 4-4 Grade Report Relation

Grade Report								
StudentID	StudentName	CampusAddress	Major	CourseID	CourseTitle	Instructor Name	Instructor Location	Grade
168300458	Williams	208 Brooks	IS	IS 350	Database Mgt	Codd	B 104	A
168300458	Williams	208 Brooks	IS	IS 465	Systems Analysis	Parsons	B 317	B
543291073	Baker	104 Phillips	Acctg	IS 350	Database Mgt	Codd	B 104	C
543291073	Baker	104 Phillips	Acctg	Acct 201	Fund Acctg	Miller	H 310	B
543291073	Baker	104 Phillips	Acctg	Mkgt 300	Intro Mktg	Bennett	B 212	A

6. **Draw a logical model for the table above, and graph and explain (with arrows) the functional dependencies in the relation.**

A logical model for the table provided can be represented using an Entity-Relationship Diagram (ERD). In this case, we have the following entities:

- Student
- Course
- Instructor

Here's how the entities relate to each other:

- Student is related to Course through Grade Report.
- Course is related to Instructor.

Let's break down the functional dependencies in the relation:

1. Student ID → Student Name, Campus Address, Major
    - The Student ID uniquely determines the student's name, campus address, and major.
2. Course ID → Course Title
    - The Course ID uniquely determines the course title.
3. Course ID → Instructor Name, Instructor Location
    - The Course ID determines the instructor's name and location since each course is taught by a specific instructor.
4. (Student ID, Course ID) → Grade
    - The combination of Student ID and Course ID uniquely determines the grade a student received in a particular course.

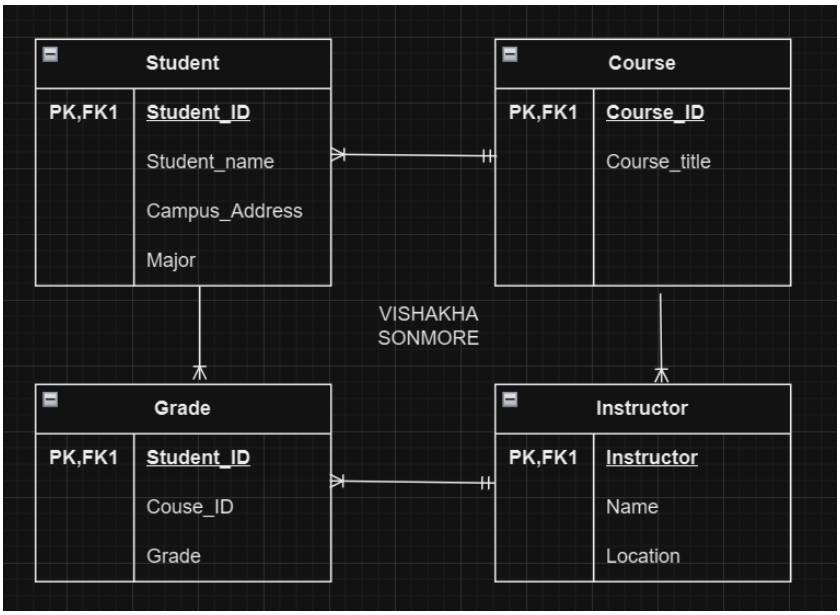
Here's the logical model represented in an ERD with the functional dependencies explained:

In the ERD:

Student and Course are related through the Grade entity, indicating the courses each student takes and the grades they receive.

Course and Instructor are related directly, showing which instructor teaches which course.

The arrows indicate the direction of the relationships and the functional dependencies between the entities.



a. In what normal form is this relation?

To determine the normal form of a relation, we need to consider the functional dependencies and the potential issues related to data redundancy and update anomalies.

Let's analyze the given relation:

1. First Normal Form (1NF):
    - All attributes contain atomic values. The table is in 1NF because each cell contains a single value.
2. Second Normal Form (2NF):
    - The table is in 2NF since there are no partial dependencies. Each non-prime attribute is fully functionally dependent on the entire primary key.
3. Third Normal Form (3NF):
    - The table is not in 3NF. There is a transitive dependency where the Instructor Name and Instructor Location depend on the Course ID, which is not part of the primary key.

To bring the relation to 3NF, we can decompose the table into two tables:

Table 1: Student_Course_Grade

Student ID	Course ID	Grade
168300458	15360	A
168300456	15465	B
543291073	$350	C
543291073	Acct 201	B
543291073	Mkt 300	A

Table 2: Course_Instructor

Course ID	Course Title	Instructor Name	Instructor Location
15360	Database Mgt	Codd	8104
15465	Systems Analysis	Parsons	8317
$350	Database Mgt	Codd	8104
Acct 201	Fund Acctg	Miller	H310
Mkt 300	Intro Mktg	Bennett	8212

In this decomposition, the Student_Course_Grade table contains information about students, courses, and grades, while the Course_Instructor table contains information about courses and instructors. This decomposition eliminates the transitive dependency, and both resulting tables are in 3NF.

b. Transform the GRADE REPORT table into relations in 3NF.

To transform the GRADE REPORT table into relations in 3NF, we need to identify and eliminate any transitive dependencies. Based on the previous analysis, we need to decompose the table into two relations:

Student_Course_Grade: This relation will contain information about students, courses, and grades.

Student ID	Course ID	Grade
168300458	15360	A
168300456	15465	B
543291073	$350	C
543291073	Acct 201	B
543291073	Mkt 300	A

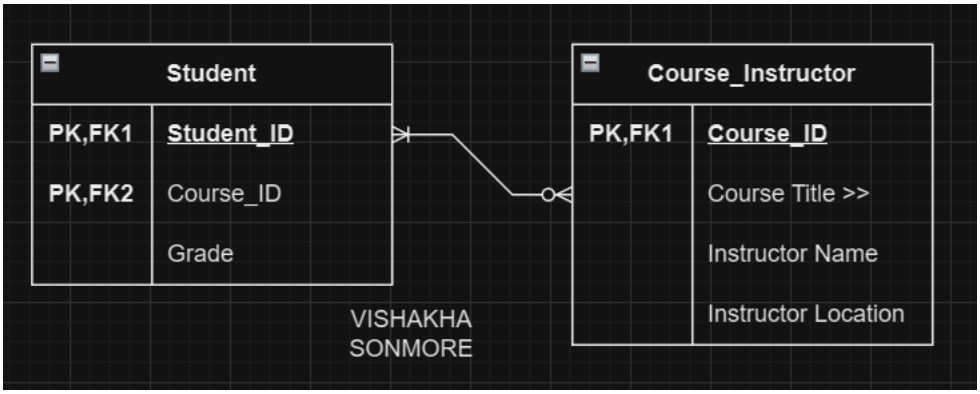
Course_Instructor: This relation will contain information about courses and instructors.

Course ID	Course Title	Instructor Name	Instructor Location
15360	Database Mgt	Codd	8104
15465	Systems Analysis	Parsons	8317
$350	Database Mgt	Codd	8104
Acct 201	Fund Acctg	Miller	H310
Mkt 300	Intro Mktg	Bennett	8212

c. Draw a model for your 3NF relations and show the primary key/foreign key relationships.

Below is an Entity-Relationship Diagram (ERD) for the two 3NF relations: Student_Course_Grade and Course_Instructor. The diagram includes primary key (PK) and foreign key (FK) relationships.

In this diagram:



- Student_Course and Course_Instructor are the two relations.
- Student ID and Course ID together form the primary key (PK) for the Student_Course relation.
- Course ID is the primary key (PK) for the Course_Instructor relation.

There is a foreign key (FK) relationship between Student_Course.Course ID (FK) and Course_Instructor.Course ID (PK), indicating the connection between the two relations.

This model reflects the relationships between students, courses, grades, and instructors in a way that adheres to the principles of 3NF.

Exercise 3: Normalizing a Shipping Manifest

Figure 4-5 shows a relation for a shipping manifest.

Shipment ID:	00-0001	Shipment Date:	01/10/2018
Origin:	Boston	Expected Arrival:	01/14/2018
Destination:	Brazil		
Ship Number:	39	Captain:	002-15
			Henry Moore

Item Number	Type	Description	Weight	Quantity	TOTALWEIGHT
3223	BM	Concrete	500	100	50,000
		Form			
3297	BM	Steel	87	2,000	174,000
		Beam			
				Shipment Total:	224,000

NOTE: This is all a single table.

a. Draw a relational schema and diagram the functional dependencies in the relation.

A relational schema is a visual representation of the structure of a database, including tables, columns, and relationships between them. In this case, you have provided two tables: one for shipment details and another for items within the shipment.

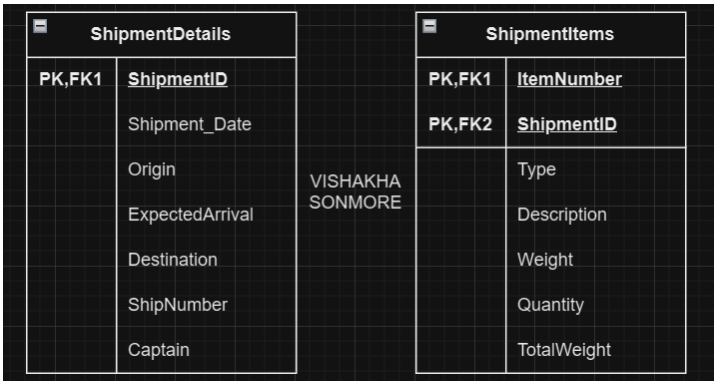
Let's represent the tables and then list the functional dependencies for each table:

Table 1: Shipment Details
- ShipmentID (Primary Key)
- ShipmentDate
- Origin
- ExpectedArrival
- Destination
- ShipNumber
- Captain

Functional Dependencies for Shipment Details:
- ShipmentID → ShipmentDate, Origin, ExpectedArrival, Destination, ShipNumber, Captain

Table 2: Shipment Items
- ItemNumber (Primary Key)
- ShipmentID (Foreign Key referencing Shipment Details)
- Type
- Description
- Weight
- Quantity
- TotalWeight

Functional Dependencies for Shipment Items:
- ItemNumber → ShipmentID, Type, Description, Weight, Quantity, TotalWeight
- ShipmentID → Type, Description, Weight, Quantity, TotalWeight



b. In what normal form is this relation?

To determine the normal form of a relation, we need to examine its functional dependencies and ensure that it meets the criteria for each normal form. Given the functional dependencies we listed earlier:

For Table 1: ShipmentDetails

- ShipmentID → ShipmentDate, Origin, ExpectedArrival, Destination, ShipNumber, Captain

For Table 2: ShipmentItems

- ItemNumber → ShipmentID, Type, Description, Weight, Quantity, TotalWeight
- ShipmentID → Type, Description, Weight, Quantity, TotalWeight

Now, let's analyze:

1. First Normal Form (1NF):
    - A relation is in 1NF if it contains only atomic values (no repeating groups or arrays).

    Both tables satisfy 1NF. Each attribute contains atomic values, and there are no repeating groups.
2. Second Normal Form (2NF):
    - A relation is in 2NF if it is in 1NF and every non-prime attribute is fully functionally dependent on the primary key.

    Both tables seem to meet the criteria for 2NF. All non-prime attributes are fully functionally dependent on the primary key.
3. Third Normal Form (3NF):
    - A relation is in 3NF if it is in 2NF and there are no transitive dependencies.

    Both tables appear to satisfy 3NF as well. There are no transitive dependencies in the given functional dependencies.

Therefore, based on the functional dependencies provided and the criteria for each normal form, the relation appears to be in Third Normal Form (3NF).

c. Decompose MANIFEST into a set of 3NF relations.

Given the functional dependencies:
For ShipmentDetails:

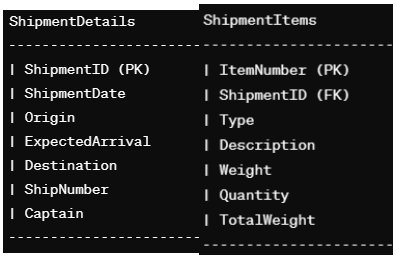
- ShipmentID → ShipmentDate, Origin, ExpectedArrival, Destination, ShipNumber, Captain

For ShipmentItems:

- ItemNumber → ShipmentID, Type, Description, Weight, Quantity, TotalWeight
- ShipmentID → Type, Description, Weight, Quantity, TotalWeight

Let's create two 3NF relations: Now, both of these relations are in 3NF. Each relation has a primary key, and there are no transitive dependencies within each relation. The ShipmentID in ShipmentItems serves as a foreign key referencing ShipmentID in ShipmentDetails, establishing a relationship between the two tables.

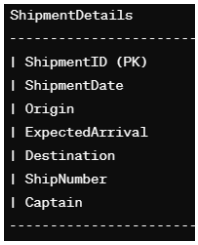
Relation 1: Shipment Details & Relation 2: ShipmentItems



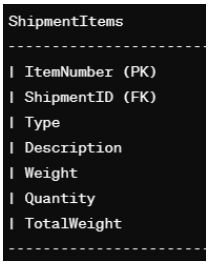
d. Draw a relational schema for your 3NF relations and show the referential integrity constraints.

Below is the relational schema for the 3NF relations, along with the referential integrity constraints:

- Relation 1: ShipmentDetails
    - Referential Integrity Constraint: None



- Relation 2: ShipmentItems
    - Referential Integrity Constraint: ShipmentID (FK) references ShipmentID (PK) in ShipmentDetails.



e. Draw your answer to part d using Microsoft Visio (or any other tool specified by your instructor).