Database Concepts Ch 2 Notes
Entity – important to the user that needs to be represented I the database.

Database products store data in the form of a relation, which is a special type of table. A relation is a two-dimensional table consisting of rows, columns that has the characteristics:

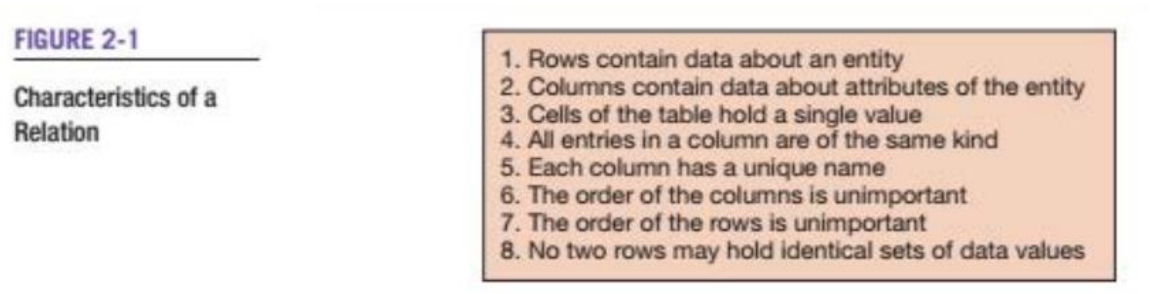


FIGURE 2-1

Characteristics of a Relation

1. Rows contain data about an entity
2. Columns contain data about attributes of the entity
3. Cells of the table hold a single value
4. All entries in a column are of the same kind
5. Each column has a unique name
6. The order of the columns is unimportant
7. The order of the rows is unimportant
8. No two rows may hold identical sets of data values

A **Key** is one or more columns of a relation that is used to identify a row; unique or nonunique.

A key that contains two or more attributes is called a **composite key**.

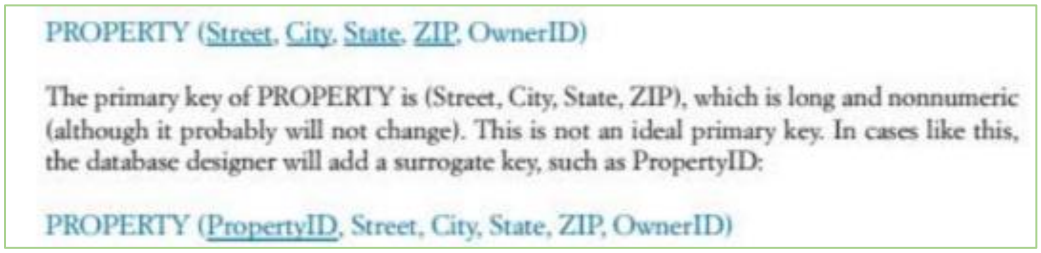
Candidate keys are keys that uniquely identify each row in a relation. Can be single-column keys or composite keys.

The **Primary Key** is the candidate key that is chosen as the key that the DBMS will use to uniquely identify each row in a relation. The PK is important because not only can it be used to identify unique rows but also because it can be used to represent rows in relationships. This key is also used by DBMS to organize storage for the relation and build index for fast retrieval of rows using PK values. –Primary Keys are indicated by underlining them. -A PK of a relation can be defined as "one or more attributes that functionally determine all the other attributes of the relation."

EX: EMPLOYEE (EmployeeNumber, FirstName, LastName, Department, EmailAddress, Phone)

PK must have unique data values inserted into every row of the table. – **Entity Integrity Constraint.**

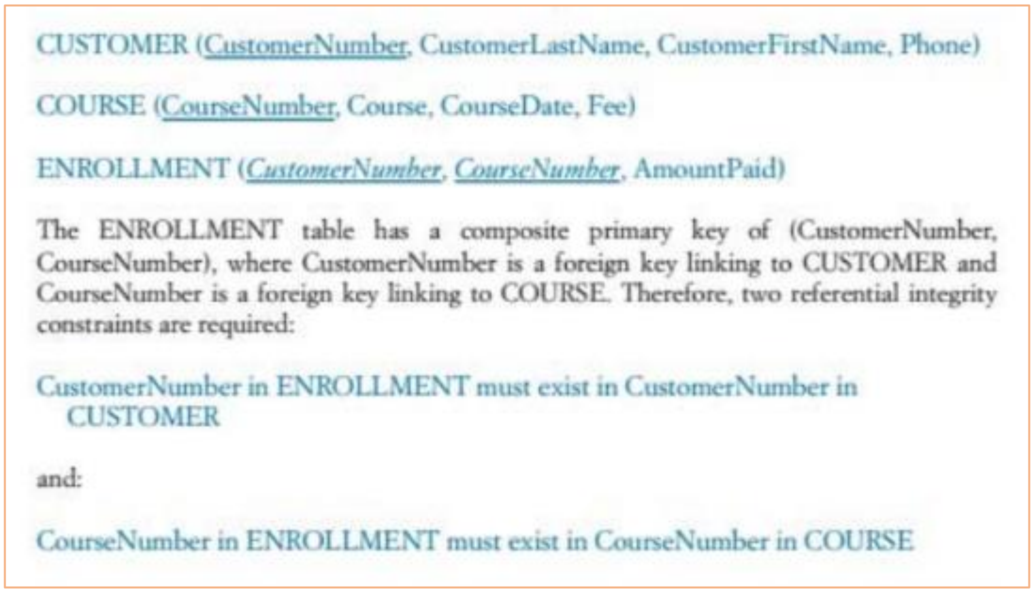
Surrogate Key is a column with a unique DBMS-assigned identifier that has been added to a table to be the primary key. The ideal PK is short and numerical and never changes.


PROPERTY (Street, City, State, ZIP, OwnerID)

The primary key of PROPERTY is (Street, City, State, ZIP), which is long and nonnumeric (although it probably will not change). This is not an ideal primary key. In cases like this, the database designer will add a surrogate key, such as PropertyID:

PROPERTY (PropertyID, Street, City, State, ZIP, OwnerID)

Foreign Key is when we place values from one relation into a second relation to represent a relationship. The values used are the Primary Key values of the first relation. *Foreign keys* are italicized.


CUSTOMER (CustomerNumber, CustomerLastName, CustomerFirstName, Phone)

COURSE (CourseNumber, Course, CourseDate, Fee)

ENROLLMENT (*CustomerNumber*, *CourseNumber*, AmountPaid)

The ENROLLMENT table has a composite primary key of (CustomerNumber, CourseNumber), where CustomerNumber is a foreign key linking to CUSTOMER and CourseNumber is a foreign key linking to COURSE. Therefore, two referential integrity constraints are required:

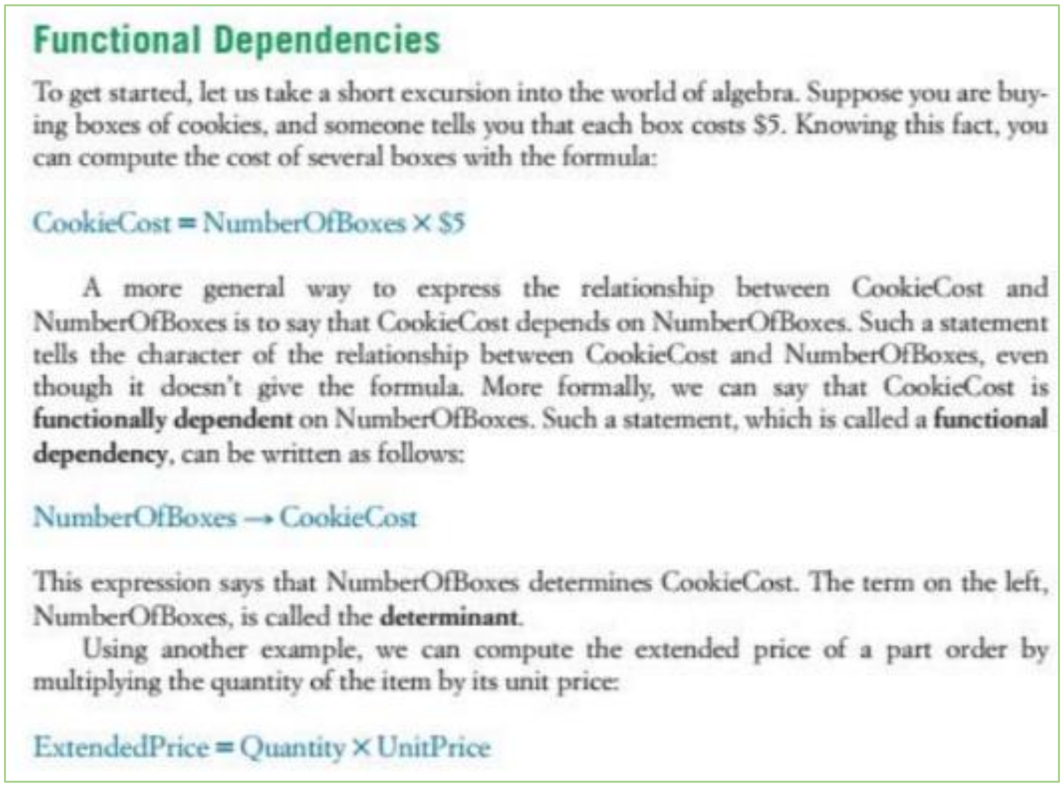
CustomerNumber in ENROLLMENT must exist in CustomerNumber in CUSTOMER

and:

CourseNumber in ENROLLMENT must exist in CourseNumber in COURSE

Note that the PK and *FK* do not need to have the same column name, the only requirement is that they have the same set of values. **Referential Integrity Constraint**. Whenever you see an *FK,* you should always look for an associated referential integrity constraint.

Database Concepts Ch 2 Notes



Functional Dependencies

To get started, let us take a short excursion into the world of algebra. Suppose you are buying boxes of cookies, and someone tells you that each box costs $5. Knowing this fact, you can compute the cost of several boxes with the formula:

CookieCost = NumberOfBoxes × $5

A more general way to express the relationship between CookieCost and NumberOfBoxes is to say that CookieCost depends on NumberOfBoxes. Such a statement tells the character of the relationship between CookieCost and NumberOfBoxes, even though it doesn't give the formula. More formally, we can say that CookieCost is **functionally dependent** on NumberOfBoxes. Such a statement, which is called a **functional dependency**, can be written as follows:

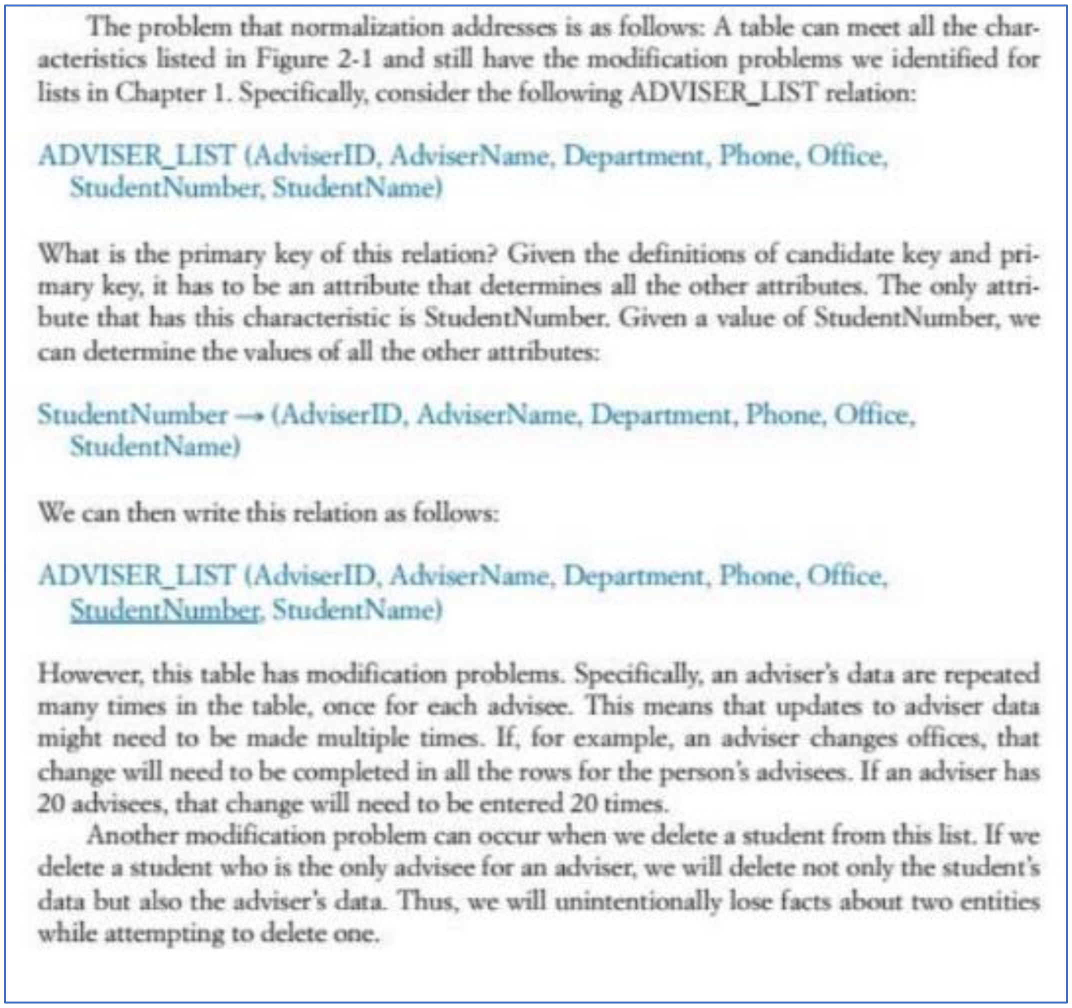
NumberOfBoxes → CookieCost

This expression says that NumberOfBoxes determines CookieCost. The term on the left, NumberOfBoxes, is called the **determinant**.

Using another example, we can compute the extended price of a part order by multiplying the quantity of the item by its unit price:

ExtendedPrice = Quantity × UnitPrice

A **null value** is a missing value in a cell in a relation.



The problem that normalization addresses is as follows: A table can meet all the characteristics listed in Figure 2-1 and still have the modification problems we identified for lists in Chapter 1. Specifically, consider the following ADVISER_LIST relation:

ADVISER_LIST (AdviserID, AdviserName, Department, Phone, Office, StudentNumber, StudentName)

What is the primary key of this relation? Given the definitions of candidate key and primary key, it has to be an attribute that determines all the other attributes. The only attribute that has this characteristic is StudentNumber. Given a value of StudentNumber, we can determine the values of all the other attributes:

StudentNumber → (AdviserID, AdviserName, Department, Phone, Office, StudentName)

We can then write this relation as follows:

ADVISER_LIST (AdviserID, AdviserName, Department, Phone, Office, StudentNumber, StudentName)

However, this table has modification problems. Specifically, an adviser's data are repeated many times in the table, once for each advisee. This means that updates to adviser data might need to be made multiple times. If, for example, an adviser changes offices, that change will need to be completed in all the rows for the person's advisees. If an adviser has 20 advisees, that change will need to be entered 20 times.

Another modification problem can occur when we delete a student from this list. If we delete a student who is the only advisee for an adviser, we will delete not only the student's data but also the adviser's data. Thus, we will unintentionally lose facts about two entities while attempting to delete one.

Normalization is the process of (or setting up steps for) breaking a table or relation with more than one theme into a set of tables such that each has only one theme.

← If you look closely at this relation, you will see a functional dependency that involves the advisor's data:

AdvisorID →(AdvisorName, Department, Phone, Office)

Now we can state the problem with this relation more accurately in terms of functional dependencies. This relation is poorly formed because it has a functional dependency that does not involve the primary key. AdvisorID is a determinant of a functional dependency, but it is not a candidate key thus cannot be a primary key under any circumstances.

Database Concepts Ch 2 Notes
<u>**Relational Design Principles:**</u>

→For a relation to be considered **well formed**, every determinant must be a candidate key.

→Any relation that is not well formed should be broken into two or more relations that are well formed.

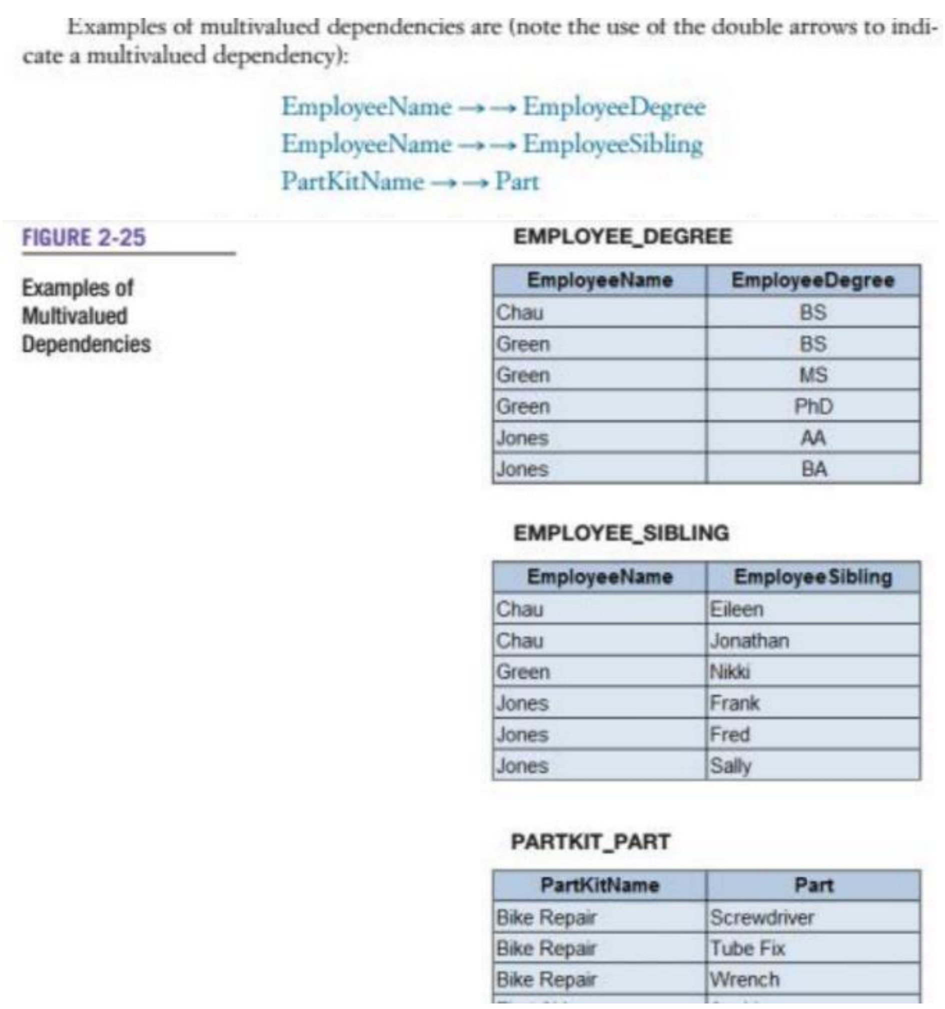
Boyce-Codd Normal Form (BCNF)

First normal form (1nf): must have the characteristics of a relation (fig 2.1 on pg. 1 of notes), have a defined <u>PK</u> and no repeating groups (or multivalued dependencies).

The Normalization Process:

- ➔ Identify all the candidate keys of the relation.
- ➔ Identify all the functional dependencies of the relation.
- ➔ Examine the determinants of functional dependencies. If any determinant is not a candidate key, the relation is not well formed, in this case:
    - o Place the columns of the functional dependency in a new relation of their own.
    - o Make the determinant off the functional dependency the <u>PK</u> of the new relation.
    - o Leave a copy of the determinant as a FK in the original relation.
    - o Create a referential integrity constraint between the original relation and new relation.
- ➔ Repeat previous step as many times necessary until every determinant of every relation is a candidate key.

When two attributes each determine one another, they are **synonyms**, they both must appear in a relation to establish their equivalent values, then the two columns are interchangeable.



Examples of multivalued dependencies are (note the use of the double arrows to indicate a multivalued dependency):

EmployeeName →→ EmployeeDegree
EmployeeName →→ EmployeeSibling
PartKitName →→ Part

FIGURE 2-25

Examples of Multivalued Dependencies

EMPLOYEE_DEGREE

EmployeeName	EmployeeDegree
Chau	BS
Green	BS
Green	MS
Green	PhD
Jones	AA
Jones	BA

EMPLOYEE_SIBLING

EmployeeName	EmployeeSibling
Chau	Eileen
Chau	Jonathan
Green	Nikki
Jones	Frank
Jones	Fred
Jones	Sally

PARTKIT_PART

PartKitName	Part
Bike Repair	Screwdriver
Bike Repair	Tube Fix
Bike Repair	Wrench

→→ **Multivalued dependency** is when modification problems are due to functional dependencies and we then normalize relations to BCNF, we eliminate **anomalies**. Multivalued dependency occurs when a determinant is matched with a particular set of values.

Unlike functional dependencies, the determinant of a multivalued dependency can never be the <u>primary</u> <u>key</u>. Multivalued dependencies pose no problem if they are in a table of their own.

When you put multivalued dependencies into a table of their own, they disappear. The result is just a table with 2 columns. When multivalued dependencies have been isolated in this way, the table is said to be in the **fourth normal form (4NF).** If A→→B then any relation that contains A and B and one additional column will have modification anomalies.

Database Concepts Ch 2 Notes

Week2 Homework Answers

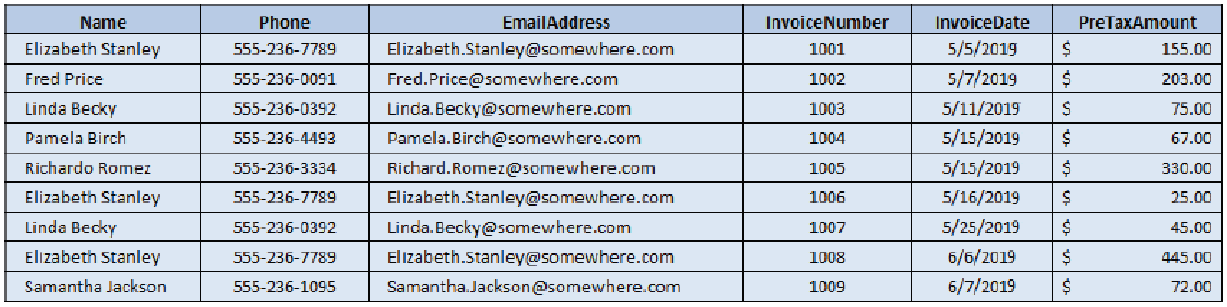
JAMES RIVER JEWELRY PROJECT QUESTIONS

James River Jewelry is a small jewelry shop. While James River Jewelry does sell typical jewelry purchased from jewelry vendors, including such items as rings, necklaces, earrings, and watches, it specializes in hard-to-find Asian jewelry. Although some Asian jewelry is manufactured jewelry purchased from vendors in the same manner as the standard jewelry is obtained, many of the Asian jewelry pieces are often unique single items purchased directly from the artisan who created the piece (the term "manufactured" would be an inappropriate description of these pieces). James River Jewelry has a small but loyal clientele, and it wants to further increase customer loyalty by creating a frequent buyer program. In this program, after every 10 purchases, a customer will receive a credit equal to 50 percent of the sum of his or her 10 most recent purchases. This credit must be applied to the next (or 11th) purchase.

Figure 2-35 shows data that James River Jewelry collects for its frequent buyer program.

FIGURE 2-35

Sample Data for James River Jewelry



Name	Phone	EmailAddress	InvoiceNumber	InvoiceDate	PreTaxAmount
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1001	5/5/2019	$ 155.00
Fred Price	555-236-0091	Fred.Price@somewhere.com	1002	5/7/2019	$ 203.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1003	5/11/2019	$ 75.00
Pamela Birch	555-236-4493	Pamela.Birch@somewhere.com	1004	5/15/2019	$ 67.00
Richardo Romez	555-236-3334	Richard.Romez@somewhere.com	1005	5/15/2019	$ 330.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1006	5/16/2019	$ 25.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1007	5/25/2019	$ 45.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1008	6/6/2019	$ 445.00
Samantha Jackson	555-236-1095	Samantha.Jackson@somewhere.com	1009	6/7/2019	$ 72.00

A. *Using these data, state assumptions about functional dependencies among the columns of data. Justify your assumptions on the basis of these sample data and also on the basis of what you know about retail sales.*

From the data it would appear:

Name → (Phone, EmailAddress)

Phone → (Name, EmailAddress)

EmailAddress → (Name, Phone)

However, these are based on a very limited dataset and cannot be trusted. For example, name is not a good determinant in a retail application; there may be many customers with the same name. It's also possible that some customers could have the same phone, even though they do not in this example.

Another functional dependency is:

InvoiceNumber → (Name, Phone, EmailAddress, InvoiceDate, PreTaxAmount)

B. *Given your assumptions in part A, comment on the appropriateness of the following designs:*

1. CUSTOMER (<u>Name</u>, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

   NOT GOOD: For example, Name does not determine InvoiceNumber because one customer may have made more than one purchase.

2. CUSTOMER (Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

   Workable, but not normalized. For example, EmailAddress → Phone. And why is InvoiceNumber the key for data about a customer? Lots of repeated data if a customer has more than one invoice.

3. CUSTOMER (Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

   GOOD FOR CUSTOMERS, BUT NOT INVOICES: Given a unique Email address, Email works as a key for customer data. Unfortunately, EmailAddress does *not* determine InvoiceNumber and therefore is not a sufficient key.

4. CUSTOMER (CustomerID, Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

   GOOD FOR CUSTOMERS, BUT NOT INVOICES: A unique ID column is a good idea, and works as a key for customer data. Unfortunately, CustomerID does *not* determine InvoiceNumber and therefore is not a sufficient key.

5. CUSTOMER (Name, Phone, EmailAddress)

   and

   PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount)

   GETTING BETTER, BUT INCOMPLETE. We cannot be sure Name is unique, and the relationship between CUSTOMER and PURCHASE is not defined.

6. CUSTOMER (Name, Phone, EmailAddress)

   and

   PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, *EmailAddress*)

   GOOD. The design breaks up the themes and has a proper foreign key. However, the use of EmailAddress as a primary key may be a problem if two customers share an Email address.

7. CUSTOMER (Name, EmailAddress)

   and

   PURCHASE (InvoiceNumber, Phone, InvoiceDate,

   PreTaxAmount, *EmailAddress*)

   A GOOD DESIGN WAS JUST MADE BAD AGAIN. The design breaks up the themes and has a proper foreign key. However, why was Phone moved? It should have been left in CUSTOMER where it will only be entered once and where:

   EmailAddress → Phone