Chapter 3

The Relational Model and Normalization

❖ REGIONAL LABS CASE QUESTIONS

Regional Labs is a company that conducts research and development work on a contract basis for other companies and organizations. Figure 3-32 shows data that Regional Labs collects about projects and the employees assigned to them. This data is stored in a relation (table) named PROJECT:

PROJECT (ProjectID, EmployeeName, EmployeeSalary)

FIGURE 3-32
Sample Data for Regional Labs

ProjectID	EmployeeName	EmployeeSalary
100-A	Eric Jones	64,000.00
100-A	Donna Smith	70,000.00
100-B	Donna Smith	70,000.00
200-A	Eric Jones	64,000.00
200-B	Eric Jones	64,000.00
200-C	Eric Parks	58,000.00
200-C	Donna Smith	70,000.00
200-D	Eric Parks	58,000.00

A. *Assuming that all functional dependencies are apparent in this data, which of the following are true?*

1. **ProjectID → EmployeeName** **FALSE**
2. **ProjectID → EmployeeSalary** **FALSE**
3. **(ProjectID, EmployeeName) → EmployeeSalary**
   TRUE, but only if **EmployeeName → EmployeeSalary**
4. **EmployeeName → EmployeeSalary** **TRUE**
5. **EmployeeSalary → ProjectID** **FALSE**
6. **EmployeeSalary → (ProjectID, EmployeeName)** **FALSE**

B. *What is the primary key of PROJECT?*

(ProjectID, EmployeeName)

C. *Are all the nonkey attributes (if any) dependent on the primary key?*

NO, EmployeeSalary is dependent only on EmployeeName

D. *In what normal form is PROJECT?*

1NF ONLY

E. *Describe two modification anomalies that affect PROJECT.*

The two modification anomalies that affect PROJECT are:

INSERTION: To give an employee a salary, we must first assign the employee to a project.

MODIFICATION: If we change a Salary, we have to change it in multiple places and may create inconsistent data.

F. *Is ProjectID a determinant? If so, based on which functional dependencies in part A?*

NO

G. *Is EmployeeName a determinant? If so, based on which functional dependencies in part A?*

YES **EmployeeName → EmployeeSalary**

H. *Is (ProjectID, EmployeeName) a determinant? If so, based on which functional dependencies in part A?*

YES **(ProjectID, EmployeeName) → EmployeeSalary**

I. *Is EmployeeSalary a determinant? If so, based on which functional dependencies in part A?*

NO Actually, for the data in Figure 3-32, it *is* a determinant. However, the dataset is ***too small*** to validate this determinant, and logically EmployeeSalary is *not* a determinant!

J. *Does this relation contain a transitive dependency? If so, what is it?*

NO

K. *Redesign the relation to eliminate modification anomalies.*

The following seems workable:

ASSIGNMENT (ProjectID, EmployeeName)

SALARY (EmployeeName, EmployeeSalary)

THE QUEEN ANNE CURIOSITY SHOP PROJECT QUESTIONS

Figure 3-33 shows typical sales data for the Queen Anne Curiosity Shop, and Figure 3-34 shows typical purchase data.

LastName	FirstName	Phone	InvoiceDate	InvoiceItem	Price	Tax	Total
Shire	Robert	206-524-2433	14-Dec-15	Antique Desk	3,000.00	249.00	3,249.00
Shire	Robert	206-524-2433	14-Dec-15	Antique Desk Chair	500.00	41.50	541.50
Goodyear	Katherine	206-524-3544	15-Dec-15	Dining Table Linens	1,000.00	83.00	1,083.00
Bancroft	Chris	425-635-9788	15-Dec-15	Candles	50.00	4.15	54.15
Griffith	John	206-524-4655	23-Dec-15	Candles	45.00	3.74	48.74
Shire	Robert	206-524-2433	5-Jan-16	Desk Lamp	250.00	20.75	270.75
Tierney	Doris	425-635-8677	10-Jan-16	Dining Table Linens	750.00	62.25	812.25
Anderson	Donna	360-538-7566	12-Jan-16	Book Shelf	250.00	20.75	270.75
Goodyear	Katherine	206-524-3544	15-Jan-16	Antique Chair	1,250.00	103.75	1,353.75
Goodyear	Katherine	206-524-3544	15-Jan-16	Antique Chair	1,750.00	145.25	1,895.25
Tierney	Doris	425-635-8677	25-Jan-16	Antique Candle Holders	350.00	29.05	379.05

FIGURE 3-33

Sample Sales Data for the Queen Anne Curiosity Shop

PurchaseItem	PurchasePrice	PurchaseDate	Vendor	Phone
Antique Desk	1,800.00	7-Nov-15	European Specialties	206-325-7866
Antique Desk	1,750.00	7-Nov-15	European Specialties	206-325-7866
Antique Candle Holders	210.00	7-Nov-15	European Specialties	206-325-7866
Antique Candle Holders	200.00	7-Nov-15	European Specialties	206-325-7866
Dining Table Linens	600.00	14-Nov-15	Linens and Things	206-325-6755
Candles	30.00	14-Nov-15	Linens and Things	206-325-6755
Desk Lamp	150.00	14-Nov-15	Lamps and Lighting	206-325-8977
Floor Lamp	300.00	14-Nov-15	Lamps and Lighting	206-325-8977
Dining Table Linens	450.00	21-Nov-15	Linens and Things	206-325-6755
Candles	27.00	21-Nov-15	Linens and Things	206-325-6755
Book Shelf	150.00	21-Nov-15	Harrison, Denise	425-746-4322
Antique Desk	1,000.00	28-Nov-15	Lee, Andrew	425-746-5433
Antique Desk Chair	300.00	28-Nov-15	Lee, Andrew	425-746-5433
Antique Chair	750.00	28-Nov-15	New York Brokerage	206-325-9088
Antique Chair	1,050.00	28-Nov-15	New York Brokerage	206-325-9088

FIGURE 3-34
Sample Purchase Data for the Queen Anne Curiosity Shop

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 sample sales data it would appear:

LastName → (FirstName, Phone)

FirstName → (LastName, Phone)

Phone → (LastName, FirstName)

Price → (Tax, Total)

(LastName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

(FirstName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

(PhoneName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

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. The one trustable functional dependency here is:

Price → (Tax, Total)

From the sample purchase data it would appear:

(InvoiceItem, PurchasePrice) → (PurchaseDate, Vendor, Phone)

(InvoiceItem, PurchasePrice, PurchaseDate) → (Vendor, Phone)

(PurchasePrice, PurchaseDate) → Invoice (InvoiceItem, Vendor, Phone)

Vendor → Phone

Phone → Vendor

However, these are based on a very limited dataset and cannot be trusted. For example, Item is not a good determinant in a retail application; there may be many Items with the same designator. It's also possible that multiple purchases on the same purchase date will be for the purchase price. The most trustworthy functional dependencies here are:

Vendor → Phone

Phone → Vendor

But, the first of these may fail if the vendor has multiple phone numbers.

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

1. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. There may be many customers with the same last name.

2. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. There may be many customers with the same last name and first name.

3. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. Phone will be fairly unique, and combined with FirstName may overcome the problem of two people with the same phone number being in the customer list (but not necessarily—what if three students are sharing an apartment, and two of them are Bill Smith and Bill Jones?). However, this will not determine the purchase information of purchase date, etc.

4. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. There may be many customers with the same last name and first name, and some of them may make a purchase on the same date. The only good thing about this is that it is starting to address the purchase information. If (LastName, FirstName) was unique, then customers would be limited to one purchase per day.

5. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. We're still trying to make the unworkable actually work. Same objections as above in 4, except that now customers would be limited to one of a particular item per day. For example, a customer could not purchase two antique chairs on the same day!

6. CUSTOMER (LastName, FirstName, Phone, Email)

   and:

   SALE (InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. Finally the customer and purchase data is effectively broken up, but there is no foreign key to link the two tables. In CUSTOMER, using (LastName, FirstName) is still a problem. In SALE, with InvoiceDate as the primary key there can only be one purchase per day!

7. CUSTOMER (LastName, FirstName, Phone, Email, *InvoiceDate*)

   and:

   SALE (InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. We've got a foreign key of PurchaseDate in SALE. But everything else that was wrong in design 6 above is still a problem. Moreover, by using PurchaseDate as the foreign key, we limit the customer to only one purchase!

8. CUSTOMER (LastName, FirstName, Phone, Email, *InvoiceDate*, *InvoiceItem*)

   and:

   SALE (InvoiceDate, Item, Price, Tax, Total)

   STILL NOT GOOD. The customer is still limited to only one purchase! However, this is the best design of the bunch if we rework the foreign keys.

C. Modify what you consider to be the best design in part B to include surrogate ID columns called CustomerID and SaleID. How does this improve the design?

The best design in part B was number 8, so we'll put in the ID columns. These columns will become the new primary keys, and we'll need to adjust the foreign key so that it is in SALE. The result is:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, InvoiceItem, Price, Tax, Total)

We now have a clean design, and CUSTOMER is in BCNF. SALE is not in BCNF because

Price → (Tax, Total)

We could further normalize SALE, but we will intentionally leave it this way. This is called **denormalization**, and is discussed in Chapter 5. The point is that creating the extra table (PRICE_TAX_TOTAL) is more trouble than it is worth. (Can you imagine what the data for that table would look like?)

The primary key problems with both tables are resolved, and now a customer can purchase as many of an item on the same date as he or she wants to!

D. Modify the design in part C by breaking SALE into two relations named SALE and SALE_ITEM. Modify columns and add additional columns as you think necessary. How does this improve the design?

The main problem with the design in part C is that only one item can be included in each sale. Moving items into a SALE_ITEM table linked to SALE will allow multiple items to be purchased as part of one sale. We'll need to include SaleID as part of a composite primary key so that the sale items are grouped according to their corresponding SALE. SaleID will also be the foreign key linking to SALE. Item and Price now belong in SALE_ITEM, and we'll need to add a PreTaxTotal to SALE—tax will now only be calculated on the pretax total value of the sale. The result is:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, PreTaxTotal, Tax, Total)

SALE_ITEM (*SaleID*, SaleItemID, InvoiceItem, Price)

We now have an improved clean design, and the CUSTOMER and SALE_ITEM tables are in BCNF. SALES is still denormalized as discussed in part C.

We have good primary and foreign keys, and now a customer can purchase as many of an item on the same date as he or she wants to and all items can be part of just one sale!

E. Given your assumptions, comment on the appropriateness of the following designs:

1. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. There may be many purchases of the same item ("Candles").

2. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. There may be many purchases of the same item that cost the same amount of money ("Candles, $30.00").

3. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. This limits purchases of a particular item to one per day.

4. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. This limits purchases of a particular item to one per vendor.

5. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate)

   and:

   VENDOR (Vendor, Phone)

   NOT GOOD. It does, however separate the two themes of PURCHASE and VENDOR. However, there is no foreign key to link the tables. Moreover, this design still limits purchases of a particular item to one per day.

6. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor)

   and:

   VENDOR (Vendor, Phone)

   BETTER, BUT STILL NOT GOOD. It separates the two themes of PURCHASE and VENDOR, but they are ***not*** properly linked by a foreign key. Note, Vendor not in italics in the PURCHASE relation.

7. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, *Vendor*)

   and:

   VENDOR (Vendor, Phone)

   GOOD, but could be BETTER. It separates the two themes of PURCHASE and VENDOR, which are now properly linked by a foreign key. However, this design still limits purchases of a particular item to one per day.

   That said, this is the best design of the bunch.

F. Modify what you consider to be the best design in part E to include surrogate ID columns called PurchaseID and VendorID. How does this improve the design?

The best design in part E was number 7, so we'll put in the ID columns. These columns will become the new primary keys, and we'll need to adjust the foreign key in PURCHASE. The result is:

PURCHASE (PurchaseID, Item, PurchasePrice, PurchaseDate, *VendorID*)

VENDOR (VendorID, Vendor, Phone)

We now have a clean design, and PURCHASE is in BCNF. VENDOR is not in BCNF because

Vendor → (VendorID, Phone)

Phone → (VendorID, Vendor)

We could further normalize VENDOR, but we will intentionally leave it this way. As discussed in part D, this is called denormalization and is discussed in Chapter 5. The point is that creating the extra table (VENDOR_PHONE) is more trouble than it is worth.

The primary key problems with both tables are resolved, and now the Queen Anne Curiosity Shop can purchase as many of an item on the same date as needed.

G. The relations in your design from part D and part F are not connected. Modify the database design so that sales data and purchase data are related.

The connection between the two parts of the database design is the item being first purchased and then sold. Thus, we can create an integrated design by replacing InvoiceItem in SALE_ITEM with PurchaseID as a foreign key. We will rename Price in SALE_ITEM as SalePrice. Our final design will be:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, PreTaxTotal, Tax, Total)

SALE_ITEM (*SaleID*, SaleItemID, *PurchaseID*, SalePrice)

PURCHASE (PurchaseID, PurchaseItem, PurchasePrice, PurchaseDate, *VendorID*)

VENDOR (VendorID, Vendor, Phone)