CHAPTER 7

Relational Database Design

Practice Exercises

7.1 Suppose that we decompose the schema $R = (A, B, C, D, E)$ into

$$(A, B, C)$$
$$(A, D, E).$$

Show that this decomposition is a lossless decomposition if the following set F of functional dependencies holds:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Answer:

A decomposition $\{R_1, R_2\}$ is a lossless decomposition if $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$. Let $R_1 = (A, B, C)$, $R_2 = (A, D, E)$, and $R_1 \cap R_2 = A$. Since A is a candidate key (see Practice Exercise 7.6), $R_1 \cap R_2 \rightarrow R_1$.

7.2 List all nontrivial functional dependencies satisfied by the relation of Figure 7.18.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Figure 7.17 Relation of Exercise 7.2.

Answer:
The nontrivial functional dependencies are: $A \rightarrow B$ and $C \rightarrow B$, and a dependency they logically imply: $AC \rightarrow B$. C does not functionally determine A because the first and third tuples have the same C but different A values. The same tuples also show B does not functionally determine A. Likewise, A does not functionally determine C because the first two tuples have the same A value and different C values. The same tuples also show B does not functionally determine C. There are 19 trivial functional dependencies of the form $\alpha \rightarrow \beta$, where $\beta \subseteq \alpha$.

7.3 Explain how functional dependencies can be used to indicate the following:

- A one-to-one relationship set exists between entity sets *student* and *instructor*.
- A many-to-one relationship set exists between entity sets *student* and *instructor*.

Answer:
Let $Pk(r)$ denote the primary key attribute of relation r.

- The functional dependencies $Pk(student) \rightarrow Pk(instructor)$ and $Pk(instructor) \rightarrow Pk(student)$ indicate a one-to-one relationship because any two tuples with the same value for *student* must have the same value for *instructor*, and any two tuples agreeing on *instructor* must have the same value for *student*.
- The functional dependency $Pk(student) \rightarrow Pk(instructor)$ indicates a many-to-one relationship since any student value which is repeated will have the same instructor value, but many student values may have the same instructor value.

7.4 Use Armstrong's axioms to prove the soundness of the union rule. (*Hint*: Use the augmentation rule to show that, if $\alpha \rightarrow \beta$, then $\alpha \rightarrow \alpha\beta$. Apply the augmentation rule again, using $\alpha \rightarrow \gamma$, and then apply the transitivity rule.)

Answer:
To prove that:

$$\text{if } \alpha \rightarrow \beta \text{ and } \alpha \rightarrow \gamma \text{ then } \alpha \rightarrow \beta\gamma$$

Following the hint, we derive:

$\alpha \rightarrow \beta$	given
$\alpha\alpha \rightarrow \alpha\beta$	augmentation rule
$\alpha \rightarrow \alpha\beta$	union of identical sets
$\alpha \rightarrow \gamma$	given
$\alpha\beta \rightarrow \gamma\beta$	augmentation rule
$\alpha \rightarrow \beta\gamma$	transitivity rule and set union commutativity

7.5 Use Armstrong's axioms to prove the soundness of the pseudotransitivity rule.

Answer:

Proof using Armstrong's axioms of the pseudotransitivity rule:
if $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$, then $\alpha\gamma \rightarrow \delta$.

$\alpha \rightarrow \beta$	given
$\alpha\gamma \rightarrow \gamma\beta$	augmentation rule and set union commutativity
$\gamma\beta \rightarrow \delta$	given
$\alpha\gamma \rightarrow \delta$	transitivity rule

7.6 Compute the closure of the following set F of functional dependencies for relation schema $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for R.

Answer:

Note: It is not reasonable to expect students to enumerate all of F^+. Some shorthand representation of the result should be acceptable as long as the nontrivial members of F^+ are found.

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$.

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, we have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCDE$	(augmentative, transitive)
Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.	

Therefore, any functional dependency with A, E, BC, or CD on the left-hand side of the arrow is in F^+, no matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A \ast\rightarrow \alpha$, $BC \ast\rightarrow \alpha$, $CD \ast\rightarrow \alpha$, $E \ast\rightarrow \alpha$ where α is any subset of $\{A, B, C, D, E\}$. The candidate keys are A, BC, CD, and E.

7.7 Using the functional dependencies of Exercise 7.6, compute the canonical cover F_c.

Answer:

The given set of FDs F is:-

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

The left side of each FD in F is unique. Also, none of the attributes in the left side or right side of any of the FDs is extraneous. Therefore the canonical cover F_c is equal to F.

7.8 Consider the algorithm in Figure 7.19 to compute α^+. Show that this algorithm is more efficient than the one presented in Figure 7.8 (Section 7.4.2) and that it computes α^+ correctly.

Answer:

The algorithm is correct because:

- If A is added to *result* then there is a proof that $\alpha \rightarrow A$. To see this, observe that $\alpha \rightarrow \alpha$ trivially, so α is correctly part of *result*. If $A \notin \alpha$ is added to *result*, there must be some FD $\beta \rightarrow \gamma$ such that $A \in \gamma$ and β is already a subset of *result*. (Otherwise *fdcount* would be nonzero and the **if** condition would be false.) A full proof can be given by induction on the depth of recursion for an execution of **addin**, but such a proof can be expected only from students with a good mathematical background.
- If $A \in \alpha^+$, then A is eventually added to *result*. We prove this by induction on the length of the proof of $\alpha \rightarrow A$ using Armstrong's axioms. First observe that if procedure **addin** is called with some argument β, all the attributes in β will be added to *result*. Also if a particular FD's *fdcount* becomes 0, all the attributes in its tail will definitely be added to *result*. The base case of the proof, $A \in \alpha \Rightarrow A \in \alpha^+$, is obviously true because the first call to **addin** has the argument α. The inductive hypothesis is that if $\alpha \rightarrow A$ can be proved in n steps or less, then $A \in$ *result*. If there is a proof in $n + 1$

```
result := ∅;
/* fdcount is an array whose ith element contains the number
   of attributes on the left side of the ith FD that are
   not yet known to be in α+ */
for i := 1 to |F| do
   begin
      let β → γ denote the ith FD;
      fdcount [i] := |β|;
   end
/* appears is an array with one entry for each attribute. The
   entry for attribute A is a list of integers. Each integer
   i on the list indicates that A appears on the left side
   of the ith FD */
for each attribute A do
   begin
      appears [A] := NIL;
      for i := 1 to |F| do
         begin
            let β → γ denote the ith FD;
            if A ∈ β then add i to appears [A];
         end
   end
addin (α);
return (result);

procedure addin (α);
for each attribute A in α do
   begin
      if A ∉ result then
         begin
            result := result ∪ {A};
            for each element i of appears[A] do
               begin
                  fdcount [i] := fdcount [i] − 1;
                  if fdcount [i] := 0 then
                     begin
                        let β → γ denote the ith FD;
                        addin (γ);
                     end
               end
         end
   end
```

Figure 7.18 An algorithm to compute α^+.

steps that $\alpha \rightarrow A$, then the last step was an application of either reflexivity, augmentation, or transitivity on a fact $\alpha \rightarrow \beta$ proved in n or fewer steps. If reflexivity or augmentation was used in the $(n+1)^{st}$ step, A must have been in *result* by the end of the n^{th} step itself. Otherwise, by the inductive hypothesis, $\beta \subseteq$ *result*. Therefore, the dependency used in proving $\beta \rightarrow \gamma$, $A \in \gamma$, will have *fdcount* set to 0 by the end of the n^{th} step. Hence A will be added to *result*.

To see that this algorithm is more efficient than the one presented in the chapter, note that we scan each FD once in the main program. The resulting array *appears* has size proportional to the size of the given FDs. The recursive calls to **addin** result in processing linear in the size of *appears*. Hence the algorithm has time complexity which is linear in the size of the given FDs. On the other hand, the algorithm given in the text has quadratic time complexity, as it may perform the loop as many times as the number of FDs, in each loop scanning all of them once.

7.9 Given the database schema $R(A, B, C)$, and a relation r on the schema R, write an SQL query to test whether the functional dependency $B \rightarrow C$ holds on relation r. Also write an SQL assertion that enforces the functional dependency. Assume that no null values are present. (Although part of the SQL standard, such assertions are not supported by any database implementation currently.)

Answer:

a. The query is given below. Its result is non-empty if and only if $B \rightarrow C$ does not hold on r.

```
select B
from r
group by B
having count(distinct C) > 1
```

b.

```
create assertion b_to_c check
   (not exists
         (select B
          from r
          group by B
          having count(distinct C) > 1
          )
   )
```

7.10 Our discussion of lossless decomposition implicitly assumed that attributes on the left-hand side of a functional dependency cannot take on null values. What could go wrong on decomposition, if this property is violated?

Answer:

The natural join operator is defined in terms of the Cartesian product and the selection operator. The selection operator gives *unknown* for any query on a null value. Thus, the natural join excludes all tuples with null values on the common attributes from the final result. Thus, the decomposition would be lossy (in a manner different from the usual case of lossy decomposition), if null values occur in the left-hand side of the functional dependency used to decompose the relation. (Null values in attributes that occur only in the right-hand side of the functional dependency do not cause any problems.)

7.11 In the BCNF decomposition algorithm, suppose you use a functional dependency $\alpha \rightarrow \beta$ to decompose a relation schema $r(\alpha, \beta, \gamma)$ into $r_1(\alpha, \beta)$ and $r_2(\alpha, \gamma)$.

a. What primary and foreign-key constraint do you expect to hold on the decomposed relations?

b. Give an example of an inconsistency that can arise due to an erroneous update, if the foreign-key constraint were not enforced on the decomposed relations above.

c. When a relation schema is decomposed into 3NF using the algorithm in Section 7.5.2, what primary and foreign-key dependencies would you expect to hold on the decomposed schema?

Answer:

a. α should be a primary key for r_1, and α should be the foreign key from r_2, referencing r_1.

b. If the foreign key constraint is not enforced, then a deletion of a tuple from r_1 would not have a corresponding deletion from the referencing tuples in r_2. Instead of deleting a tuple from r, this would amount to simply setting the value of α to null in some tuples.

c. For every schema $r_i(\alpha\beta)$ added to the decomposition because of a functional dependency $\alpha \rightarrow \beta$, α should be made the primary key. Also, a candidate key γ for the original relation is located in some newly created relation r_k and is a primary key for that relation.
Foreign-key constraints are created as follows: for each relation r_i created above, if the primary key attributes of r_i also occur in any other relation r_j, then a foreign-key constraint is created from those attributes in r_j, referencing (the primary key of) r_i.

7.12 Let $R_1, R_2, \ldots, R_n$ be a decomposition of schema U. Let $u(U)$ be a relation, and let $r_i = \Pi_{R_i}(u)$. Show that

$$u \subseteq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

Answer:

Consider some tuple t in u.

Note that $r_i = \Pi_{R_i}(u)$ implies that $t[R_i] \in r_i$, $1 \leq i \leq n$. Thus,

$$t[R_1] \bowtie t[R_2] \bowtie \ldots \bowtie t[R_n] \in r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

By the definition of natural join,

$$t[R_1] \bowtie t[R_2] \bowtie \ldots \bowtie t[R_n] = \Pi_\alpha(\sigma_\beta(t[R_1] \times t[R_2] \times \ldots \times t[R_n]))$$

where the condition β is satisfied if values of attributes with the same name in a tuple are equal and where $\alpha = U$. The Cartesian product of single tuples generates one tuple. The selection process is satisfied because all attributes with the same name must have the same value since they are projections from the same tuple. Finally, the projection clause removes duplicate attribute names.

By the definition of decomposition, $U = R_1 \cup R_2 \cup \ldots \cup R_n$, which means that all attributes of t are in $t[R_1] \bowtie t[R_2] \bowtie \ldots \bowtie t[R_n]$. That is, t is equal to the result of this join.

Since t is any arbitrary tuple in u,

$$u \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

7.13 Show that the decomposition in Exercise 7.1 is not a dependency-preserving decomposition.

Answer:

Therer are several functional dependencies that are not preserved. We discuss one example here. The dependency $B \rightarrow D$ is not preserved. F_1, the restriction of F to (A, B, C) is $A \rightarrow ABC$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow A$, $B \rightarrow B$, $C \rightarrow C$, $AB \rightarrow AC$, $AB \rightarrow ABC$, $AB \rightarrow BC$, $AB \rightarrow AB$, $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow C$, AC (same as AB), BC (same as AB), ABC (same as AB). F_2, the restriction of F to (C, D, E) is $A \rightarrow ADE$, $A \rightarrow AD$, $A \rightarrow AE$, $A \rightarrow DE$, $A \rightarrow A$, $A \rightarrow D$, $A \rightarrow E$, $D \rightarrow D$, E (same as A), AD, AE, DE, ADE (same as A). $(F_1 \cup F_2)^+$ is easily seen not to contain $B \rightarrow D$ since the only FD in $F_1 \cup F_2$ with B as the left side is $B \rightarrow B$, a trivial FD. Thus $B \rightarrow D$ is not preserved.

A simpler argument is as follows: F_1 contains no dependencies with D on the right side of the arrow. F_2 contains no dependencies with B on the left side of the arrow. Therefore for $B \rightarrow D$ to be preserved there must be a functional dependency $B \rightarrow \alpha$ in F_1^+ and $\alpha \rightarrow D$ in F_2^+ (so $B \rightarrow D$ would follow by

transitivity). Since the intersection of the two schemes is A, $\alpha = A$. Observe that $B \rightarrow A$ is not in F_1^+ since $B^+ = BD$.

7.14 Show that there can be more than one canonical cover for a given set of functional dependencies, using the following set of dependencies:

$$X \rightarrow YZ,\ Y \rightarrow XZ, \text{ and } Z \rightarrow XY.$$

Answer: Consider the first functional dependency. We can verify that Z is extraneous in $X \rightarrow YZ$ and delete it. Subsequently, we can similarly check that X is extraneous in $Y \rightarrow XZ$ and delete it, and that Y is extraneous in $Z \rightarrow XY$ and delete it, resulting in a canonical cover $X \rightarrow Y, Y \rightarrow Z, Z \rightarrow X$.
However, we can also verify that Y is extraneous in $X \rightarrow YZ$ and delete it. Subsequently, we can similarly check that Z is extraneous in $Y \rightarrow XZ$ and delete it, and that X is extraneous in $Z \rightarrow XY$ and delete it, resulting in a canonical cover $X \rightarrow Z, Y \rightarrow X, Z \rightarrow Y$.

7.15 The algorithm to generate a canonical cover only removes one extraneous attribute at a time. Use the functional dependencies from Exercise 7.14 to show what can go wrong if two attributes inferred to be extraneous are deleted at once.

Answer: In $X \rightarrow YZ$, one can infer that Y is extraneous, and so is Z. But deleting both will result in a set of dependencies from which $X \rightarrow YZ$ can no longer be inferred. Deleting Y results in Z no longer being extraneous, and deleting Z results in Y no longer being extraneous. The canonical cover algorithm only deletes one attribute at a time, avoiding the problem that could occur if two attributes are deleted at the same time.

7.16 Show that it is possible to ensure that a dependency-preserving decomposition into 3NF is a lossless decomposition by guaranteeing that at least one schema contains a candidate key for the schema being decomposed. (*Hint*: Show that the join of all the projections onto the schemas of the decomposition cannot have more tuples than the original relation.)

Answer:
Let F be a set of functional dependencies that hold on a schema R. Let $\sigma = \{R_1, R_2, \ldots, R_n\}$ be a dependency-preserving 3NF decomposition of R. Let X be a candidate key for R.
Consider a legal instance r of R. Let $j = \Pi_X(r) \bowtie \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \ldots \bowtie \Pi_{R_n}(r)$. We want to prove that $r = j$.

We claim that if t_1 and t_2 are two tuples in j such that $t_1[X] = t_2[X]$, then $t_1 = t_2$. To prove this claim, we use the following inductive argument:
Let $F' = F_1 \cup F_2 \cup \ldots \cup F_n$, where each F_i is the restriction of F to the schema R_i in σ. Consider the use of the algorithm given in Figure 7.8 to compute the

closure of X under F'. We use induction on the number of times that the *for* loop in this algorithm is executed.

- *Basis*: In the first step of the algorithm, *result* is assigned to X, and hence given that $t_1[X] = t_2[X]$, we know that $t_1[result] = t_2[result]$ is true.
- *Induction Step*: Let $t_1[result] = t_2[result]$ be true at the end of the k th execution of the *for* loop.

  Suppose the functional dependency considered in the $k+1$ th execution of the *for* loop is $\beta \rightarrow \gamma$, and that $\beta \subseteq result$. $\beta \subseteq result$ implies that $t_1[\beta] = t_2[\beta]$ is true. The facts that $\beta \rightarrow \gamma$ holds for some attribute set R_i in σ and that $t_1[R_i]$ and $t_2[R_i]$ are in $\Pi_{R_i}(r)$ imply that $t_1[\gamma] = t_2[\gamma]$ is also true. Since γ is now added to *result* by the algorithm, we know that $t_1[result] = t_2[result]$ is true at the end of the $k + 1$ th execution of the *for* loop.

Since σ is dependency-preserving and X is a key for R, all attributes in R are in *result* when the algorithm terminates. Thus, $t_1[R] = t_2[R]$ is true, that is, $t_1 = t_2$ – as claimed earlier.

Our claim implies that the size of $\Pi_X(j)$ is equal to the size of j. Note also that $\Pi_X(j) = \Pi_X(r) = r$ (since X is a key for R). Thus we have proved that the size of j equals that of r. Using the result of Exercise 7.12, we know that $r \subseteq j$. Hence we conclude that $r = j$.

Note that since X is trivially in 3NF, $\sigma \cup \{X\}$ is a dependency-preserving lossless decomposition into 3NF.

7.17 Give an example of a relation schema R' and set F' of functional dependencies such that there are at least three distinct lossless decompositions of R' into BCNF.

Answer:

Given the relation $R' = (A, B, C, D)$ the set of functional dependencies $F' = A \rightarrow B, C \rightarrow D, B \rightarrow C$ allows three distinct BCNF decompositions.

$$R_1 = \{(A, B), (C, D), (B, C)\}$$

is in BCNF as is

$$R_2 = \{(A, B), (C, D), (A, C)\}$$

$$R_3 = \{(B, C), (A, D), (A, B)\}$$

7.18 Let a **prime** attribute be one that appears in at least one candidate key. Let α and β be sets of attributes such that $\alpha \rightarrow \beta$ holds, but $\beta \rightarrow \alpha$ does not hold. Let A be

an attribute that is not in α, is not in β, and for which $\beta \rightarrow A$ holds. We say that A is **transitively dependent** on α. We can restate the definition of 3NF as follows: A relation schema R is in 3NF with respect to a set F of functional dependencies if there are no nonprime attributes A in R for which A is transitively dependent on a key for R. Show that this new definition is equivalent to the original one.

Answer:

Suppose R is in 3NF according to the textbook definition. We show that it is in 3NF according to the definition in the exercise. Let A be a nonprime attribute in R that is transitively dependent on a key α for R. Then there exists $\beta \subseteq R$ such that $\beta \rightarrow A$, $\alpha \rightarrow \beta$, $A \notin \alpha$, $A \notin \beta$, and $\beta \rightarrow \alpha$ does not hold. But then $\beta \rightarrow A$ violates the textbook definition of 3NF since

- $A \notin \beta$ implies $\beta \rightarrow A$ is nontrivial
- Since $\beta \rightarrow \alpha$ does not hold, β is not a superkey
- A is not any candidate key, since A is nonprime

Now we show that if R is in 3NF according to the exercise definition, it is in 3NF according to the textbook definition. Suppose R is not in 3NF according to the the textbook definition. Then there is an FD $\alpha \rightarrow \beta$ that fails all three conditions. Thus

- $\alpha \rightarrow \beta$ is nontrivial.
- α is not a superkey for R.
- Some A in $\beta - \alpha$ is not in any candidate key.

This implies that A is nonprime and $\alpha \rightarrow A$. Let γ be a candidate key for R. Then $\gamma \rightarrow \alpha$, $\alpha \rightarrow \gamma$ does not hold (since α is not a superkey), $A \notin \alpha$, and $A \notin \gamma$ (since A is nonprime). Thus A is transitively dependent on γ, violating the exercise definition.

7.19 A functional dependency $\alpha \rightarrow \beta$ is called a **partial dependency** if there is a proper subset γ of α such that $\gamma \rightarrow \beta$; we say that β is *partially dependent* on α. A relation schema R is in **second normal form** (**2NF**) if each attribute A in R meets one of the following criteria:

- It appears in a candidate key.
- It is not partially dependent on a candidate key.

Show that every 3NF schema is in 2NF. (*Hint*: Show that every partial dependency is a transitive dependency.)

Answer:

Referring to the definitions in Exercise 7.18, a relation schema R is said to be in 3NF if there is no nonprime attribute A in R for which A is transitively dependent on a key for R.

We can also rewrite the definition of 2NF given here as:
"A relation schema R is in 2NF if no nonprime attribute A is partially dependent on any candidate key for R."

To prove that every 3NF schema is in 2NF, it suffices to show that if a nonprime attribute A is partially dependent on a candidate key α, then A is also transitively dependent on the key α.

Let A be a nonprime attribute in R. Let α be a candidate key for R. Suppose A is partially dependent on α.

- From the definition of a partial dependency, we know that for some proper subset β of α, $\beta \rightarrow A$.
- Since $\beta \subset \alpha$, $\alpha \rightarrow \beta$. Also, $\beta \rightarrow \alpha$ does not hold, since α is a candidate key.
- Finally, since A is nonprime, it cannot be in either β or α.

Thus we conclude that $\alpha \rightarrow A$ is a transitive dependency. Hence we have proved that every 3NF schema is also in 2NF.

7.20 Give an example of a relation schema R and a set of dependencies such that R is in BCNF but is not in 4NF.

Answer:
There are, of course, an infinite number of such examples. We show the simplest one here.
Let R be the schema (A, B, C) with the only nontrivial dependency being $A \twoheadrightarrow B$