Exercises

7.1 Explain what is meant by *repetition of information* and *inability to represent information*. Explain why each of these properties may indicate a bad relational-database design.

Answer:

- Repetition of information is a condition in a relational database where the values of one attribute are determined by the values of another attribute in the same relation, and both values are repeated throughout the relation. This is a bad relational database design because it increases the storage required for the relation and it makes updating the relation more difficult.
- Inability to represent information is a condition where a relationship exists among only a proper subset of the attributes in a relation. This is bad relational database design because all the unrelated attributes must be filled with null values otherwise a tuple without the unrelated information cannot be inserted into the relation.
- Loss of information is a condition of a relational database which results from the decomposition of one relation into two relations and which cannot be combined to recreate the original relation. It is a bad relational database design because certain queries cannot be answered using the reconstructed relation that could have been answered using the original relation.

7.2 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-join 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-join 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 Exercise 7.11), Therefore $R_1 \cap R_2 \rightarrow R_1$.

7.3 Why are certain functional dependencies called *trivial* functional dependencies?

Answer: Certain functional dependencies are called trivial functional dependencies because they are satisfied by all relations.

7.4 List all functional dependencies satisfied by the relation of Figure 7.21.

Answer: The nontrivial functional dependencies are: $A \rightarrow B$ and $C \rightarrow B$,

and a dependency they logically imply: $AC \rightarrow B$. There are 19 trivial functional dependencies of the form $\alpha \rightarrow \beta$, where $\beta \subseteq \alpha$. 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.

7.5 Use the definition of functional dependency to argue that each of Armstrong's axioms (reflexivity, augmentation, and transitivity) is sound.

Answer: The definition of functional dependency is: $\alpha \rightarrow \beta$ holds on R if in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.

Reflexivity rule: if α is a set of attributes, and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$.
Assume $\exists\, t_1$ and t_2 such that $t_1[\alpha] = t_2[\alpha]$

$t_1[\beta] = t_2[\beta]$	since $\beta \subseteq \alpha$
$\alpha \rightarrow \beta$	definition of FD

Augmentation rule: if $\alpha \rightarrow \beta$, and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$.
Assume $\exists\, t_1,\ t_2$ such that $t_1[\gamma\alpha] = t_2[\gamma\alpha]$

$t_1[\gamma] = t_2[\gamma]$	$\gamma \subseteq \gamma\alpha$
$t_1[\alpha] = t_2[\alpha]$	$\alpha \subseteq \gamma\alpha$
$t_1[\beta] = t_2[\beta]$	definition of $\alpha \rightarrow \beta$
$t_1[\gamma\beta] = t_2[\gamma\beta]$	$\gamma\beta = \gamma \cup \beta$
$\gamma\alpha \rightarrow \gamma\beta$	definition of FD

Transitivity rule: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$.
Assume $\exists\, t_1,\ t_2$ such that $t_1[\alpha] = t_2[\alpha]$

$t_1[\beta] = t_2[\beta]$	definition of $\alpha \rightarrow \beta$
$t_1[\gamma] = t_2[\gamma]$	definition of $\beta \rightarrow \gamma$
$\alpha \rightarrow \gamma$	definition of FD

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

- A one-to-one relationship set exists between entity sets *account* and *customer*.
- A many-to-one relationship set exists between entity sets *account* and *customer*.

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 7.21. Relation of Exercise 7.4.

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

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

7.7 Consider the following proposed rule for functional dependencies: If $\alpha \rightarrow \beta$ and $\gamma \rightarrow \beta$, then $\alpha \rightarrow \gamma$. Prove that this rule is *not* sound by showing a relation *r* that satisfies $\alpha \rightarrow \beta$ and $\gamma \rightarrow \beta$, but does not satisfy $\alpha \rightarrow \gamma$.
Answer: Consider the following rule: if $A \rightarrow B$ and $C \rightarrow B$, then $A \rightarrow C$. That is, $\alpha = A$, $\beta = B$, $\gamma = C$. The following relation r is a counterexample to the rule.

r:

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2

Note: $A \rightarrow B$ and $C \rightarrow B$, (since no 2 tuples have the same C value, $C \rightarrow B$ is true trivially). However, it is not the case that $A \rightarrow C$ since the same A value is in two tuples, but the C value in those tuples disagree.

7.8 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

7.9 Use Armstrong's axioms to prove the soundness of the decomposition rule.
Answer: The decomposition rule, and its derivation from Armstrong's axioms are given below:

if $\alpha \rightarrow \beta\gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.

$\alpha \rightarrow \beta\gamma$	given
$\beta\gamma \rightarrow \beta$	reflexivity rule
$\alpha \rightarrow \beta$	transitivity rule
$\beta\gamma \rightarrow \gamma$	reflexive rule
$\alpha \rightarrow \gamma$	transitive rule

7.10 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

7.11 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: 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*.
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 $A* \rightarrow \alpha$, $BC* \rightarrow \alpha$, $CD* \rightarrow \alpha$, $E* \rightarrow \alpha$ where α is any subset of $\{A,\ B,\ C,\ D,\ E\}$. The candidate keys are $A,\ BC,\ CD$, and E.

7.12 Using the functional dependencies of Exercise 7.11, compute B^+.
Answer: Computing B^+ by the algorithm in Figure 7.7 we start with $result = \{B\}$. Considering FDs of the form $\beta \rightarrow \gamma$ in F, we find that the only dependencies satisfying $\beta \subseteq result$ are $B \rightarrow B$ and $B \rightarrow D$. Therefore $result = \{B,\ D\}$. No more dependencies in F apply now. Therefore $B^+ = \{B,\ D\}$

7.13 Using the functional dependencies of Exercise 7.11, 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.

7.14 Consider the algorithm in Figure 7.22 to compute α^+. Show that this algorithm is more efficient than the one presented in Figure 7.7 (Section 7.3.3) 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 hypotheses 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 7.22. 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.

7.15 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.

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
        )
    )
```

7.16 Show that the following decomposition of the schema R of Exercise 7.2 is not a lossless-join decomposition:

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

Hint: Give an example of a relation r on schema R such that

$$\Pi_{A,\,B,\,C}\,(r)\ \bowtie\ \Pi_{C,\,D,\,E}\,(r)\ \neq\ r$$

Answer: Following the hint, use the following example of r:

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

With $R_1\ =\ (A,\ B,\ C)$, $R_2\ =\ (C,\ D,\ E)$:

a. $\Pi_{R_1}\,(r)$ would be:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_1

b. $\Pi_{R_2}\,(r)$ would be:

C	D	E
c_1	d_1	e_1
c_1	d_2	e_2

c. $\Pi_{R_1}\,(r)\ \bowtie\ \Pi_{R_2}\,(r)$ would be:

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

Clearly, $\Pi_{R_1}\,(r)\ \bowtie\ \Pi_{R_2}\,(r)\ \neq\ r$. Therefore, this is a lossy join.

7.17 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,

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

7.18 Show that the decomposition in Exercise 7.2 is not a dependency-preserving decomposition.

Answer: 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. We shall see in Exercise 7.22 that $B \rightarrow D$ is indeed in F^+. Thus $B \rightarrow D$ is not preserved. Note that $CD \rightarrow ABCDE$ is also 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 an FD $B \rightarrow \alpha$ 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 $B^+ = BD$.

7.19 Show that it is possible to ensure that a dependency-preserving decomposition into 3NF is a lossless-join 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 7.7 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 7.17, 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-join decomposition into 3NF.

7.20 List the three design goals for relational databases, and explain why each is desirable.
Answer: The three design goals are lossless-join decompositions, dependency preserving decompositions, and minimization of repetition of information. They are desirable so we can maintain an accurate database, check correctness of updates quickly, and use the smallest amount of space possible.

7.21 Give a lossless-join decomposition into BCNF of schema R of Exercise 7.2.
Answer: From Exercise 7.11, we know that $B \rightarrow D$ is nontrivial and the left hand side is not a superkey. By the algorithm of Figure 7.13 we derive the relations $\{(A, B, C, E), (B, D)\}$. This is in BCNF.

7.22 Give an example of a relation schema R' and set F' of functional dependencies such that there are at least three distinct lossless-join 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

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

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

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

7.23 In designing a relational database, why might we choose a non-BCNF design?
Answer: BCNF is not always dependency preserving. Therefore, we may want to choose another normal form (specifically, 3NF) in order to make checking dependencies easier during updates. This would avoid joins to check dependencies and increase system performance.

7.24 Give a lossless-join, dependency-preserving decomposition into 3NF of schema R of Exercise 7.2.
Answer: First we note that the dependencies given in Exercise 7.2 form a canonical cover. Generating the schema from the algorithm of Figure 7.14 we get

$$R' = \{(A, B, C), (C, D, E), (B, D), (E, A)\}.$$

Schema $(A,\ B,\ C)$ contains a candidate key. Therefore R' is a third normal form dependency-preserving lossless-join decomposition.

Note that the original schema $R = (A,\ B,\ C,\ D,\ E)$ is already in 3NF. Thus, it was not necessary to apply the algorithm as we have done above. The single original schema is trivially a lossless join, dependency-preserving decomposition.

7.25 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 our 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 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