19

SCHEMA REFINEMENT AND NORMAL FORMS

Exercise 19.1 Briefly answer the following questions:

1. Define the term *functional dependency.*
2. Why are some functional dependencies called *trivial*?
3. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key AB under which R is in 1NF but not in 2NF.
4. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key AB under which R is in 2NF but not in 3NF.
5. Consider the relation schema *R(A,B,C)*, which has the FD $B \rightarrow C$. If A is a candidate key for R, is it possible for R to be in BCNF? If so, under what conditions? If not, explain why not.
6. Suppose we have a relation schema *R(A,B,C)* representing a relationship between two entity sets with keys A and B, respectively, and suppose that R has (among others) the FDs $A \rightarrow B$ and $B \rightarrow A$. Explain what such a pair of dependencies means (i.e., what they imply about the relationship that the relation models).

Answer 19.1

1. Let R be a relational schema and let X and Y be two subsets of the set of all attributes of R. We say Y is functionally dependent on X, written X → Y, if the Y-values are determined by the X-values. More precisely, for any two tuples r_1 and r_2 in (any instance of) R

$$\pi_X(r_1) = \pi_X(r_2) \quad \Rightarrow \quad \pi_Y(r_1) = \pi_Y(r_2)$$

2. Some functional dependencies are considered trivial because they contain superfluous attributes that do not need to be listed. Consider the FD: $A \rightarrow AB$. By reflexivity, A always implies A, so that the A on the right hand side is not necessary and can be dropped. The proper form, without the trivial dependency would then be $A \rightarrow B$.

3. Consider the set of FD: $AB \rightarrow CD$ and $B \rightarrow C$. AB is obviously a key for this relation since $AB \rightarrow CD$ implies $AB \rightarrow ABCD$. It is a primary key since there are no smaller subsets of keys that hold over *R(A,B,C,D)*. The FD: $B \rightarrow C$ violates 2NF since:
   - $C \in B$ is false; that is, it *is not* a trivial FD
   - B *is not* a superkey
   - C *is not* part of some key for R
   - B *is* a proper subset of the key AB (transitive dependency)

4. Consider the set of FD: $AB \rightarrow CD$ and $C \rightarrow D$. AB is obviously a key for this relation since $AB \rightarrow CD$ implies $AB \rightarrow ABCD$. It is a primary key since there are no smaller subsets of keys that hold over *R(A,B,C,D)*. The FD: $C \rightarrow D$ violates 3NF but not 2NF since:
   - $D \in C$ is false; that is, it *is not* a trivial FD
   - C *is not* a superkey
   - D *is not* part of some key for R

5. The only way R could be in BCNF is if B includes a key, *i.e.* B is a key for R.

6. It means that the relationship is one to one. That is, each A entity corresponds to at most one B entity and vice-versa. (In addition, we have the dependency $AB \rightarrow C$, from the semantics of a relationship set.)

Exercise 19.2 Consider a relation R with five attributes $ABCDE$. You are given the following dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$.

1. List all keys for R.
2. Is R in 3NF?
3. Is R in BCNF?

Answer 19.2

1. CDE, ACD, BCD
2. R is in 3NF because B, E and A are all parts of keys.
3. R is not in BCNF because none of A, BC and ED contain a key.

Exercise 19.3 Consider the relation shown in Figure 19.1.

1. List all the functional dependencies that this relation instance satisfies.

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_2	y_1	z_1
x_2	y_1	z_3

Figure 19.1 Relation for Exercise 19.3.

2. Assume that the value of attribute Z of the last record in the relation is changed from z_3 to z_2. Now list all the functional dependencies that this relation instance satisfies.

Answer 19.3

1. The following functional dependencies hold over R: $Z \rightarrow Y$, $X \rightarrow Y$, and $XZ \rightarrow Y$
2. Same as part 1. Functional dependency set is unchanged.

Exercise 19.4 Assume that you are given a relation with attributes $ABCD$.

1. Assume that no record has NULL values. Write an SQL query that checks whether the functional dependency $A \rightarrow B$ holds.
2. Assume again that no record has NULL values. Write an SQL assertion that enforces the functional dependency $A \rightarrow B$.
3. Let us now assume that records could have NULL values. Repeat the previous two questions under this assumption.

Answer 19.4 Assuming...

1. The following statement returns 0 iff no statement violates the FD $A \rightarrow B$.

```
SELECT COUNT (*)
FROM   R AS R1, R AS R2
WHERE  (R1.B != R2.B) AND (R1.A = R2.A)
```

2.
```
CREATE ASSERTION ADeterminesB
CHECK  ((SELECT COUNT (*)
         FROM   R AS R1, R AS R2
         WHERE  (R1.B != R2.B) AND (R1.A = R2.A))
         =0)
```

3. Note that the following queries can be written with the NULL and NOT NULL interchanged. Since we are doing a full join of a table and itself, we are creating tuples in sets of two therefore the order is not important.

```
SELECT COUNT (*)
FROM   R AS R1, R AS R2
WHERE  ((R1.B != R2.B) AND (R1.A = R2.A))
       OR ((R1.B is NULL) AND (R2.B is NOT NULL)
              AND (R1.A = R2.A))

CREATE ASSERTION ADeterminesBNull
CHECK  ((SELECT COUNT (*)
       FROM   R AS R1, R AS R2
       WHERE  ((R1.B != R2.B) AND (R1.A = R2.A)))
              OR ((R1.B is NULL) AND (R2.B is NOT NULL)
                     AND (R1.A = R2.A))
       =0)
```

Exercise 19.5 Consider the following collection of relations and dependencies. Assume that each relation is obtained through decomposition from a relation with attributes *ABCDEFGHI* and that all the known dependencies over relation *ABCDEFGHI* are listed for each question. (The questions are independent of each other, obviously, since the given dependencies over *ABCDEFGHI* are different.) For each (sub)relation: (a) State the strongest normal form that the relation is in. (b) If it is not in BCNF, decompose it into a collection of BCNF relations.

1. *R1(A,C,B,D,E)*, $A \rightarrow B$, $C \rightarrow D$
2. *R2(A,B,F)*, $AC \rightarrow E$, $B \rightarrow F$
3. *R3(A,D)*, $D \rightarrow G$, $G \rightarrow H$
4. *R4(D,C,H,G)*, $A \rightarrow I$, $I \rightarrow A$
5. *R5(A,I,C,E)*

Answer 19.5

1. 1NF. BCNF decomposition: AB, CD, ACE.
2. 1NF. BCNF decomposition: AB, BF
3. BCNF.
4. BCNF.
5. BCNF.

Exercise 19.6 Suppose that we have the following three tuples in a legal instance of a relation schema S with three attributes ABC (listed in order): (1,2,3), (4,2,3), and (5,3,3).

1. Which of the following dependencies can you infer does *not* hold over schema S?

   (a) $A \rightarrow B$, (b) $BC \rightarrow A$, (c) $B \rightarrow C$

2. Can you identify any dependencies that hold over S?

Answer 19.6

1. $BC\rightarrow A$ does not hold over S (look at the tuples (1,2,3) and (4,2,3)). The other tuples hold over S.

2. No. Given just an instance of S, we can say that certain dependencies (e.g., $A \rightarrow B$ and B $\rightarrow C$) are not violated by this instance, but we cannot say that these dependencies hold with respect to S. To say that an FD holds w.r.t. a relation is to make a statement about *all* allowable instances of that relation!

Exercise 19.7 Suppose you are given a relation R with four attributes $ABCD$. For each of the following sets of FDs, assuming those are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) Identify the best normal form that R satisfies (1NF, 2NF, 3NF, or BCNF). (c) If R is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies.

1. $C \rightarrow D$, $C \rightarrow A$, $B \rightarrow C$
2. $B \rightarrow C$, $D \rightarrow A$
3. $ABC \rightarrow D$, $D \rightarrow A$
4. $A \rightarrow B$, $BC \rightarrow D$, $A \rightarrow C$
5. $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$

Answer 19.7

1. (a) Candidate keys: B

   (b) R is in 2NF but not 3NF.

   (c) $C \rightarrow D$ and $C \rightarrow A$ both cause violations of BCNF. One way to obtain a (lossless) join preserving decomposition is to decompose R into AC, BC, and CD.

2. (a) Candidate keys: BD

   (b) R is in 1NF but not 2NF.

   (c) Both $B \rightarrow C$ and $D \rightarrow A$ cause BCNF violations. The decomposition: *AD*, *BC*, *BD* (obtained by first decomposing to *AD*, *BCD*) is BCNF and lossless and join-preserving.

3. (a) Candidate keys: *ABC*, *BCD*
   (b) *R* is in 3NF but not BCNF.
   (c) *ABCD* is not in BCNF since $D \rightarrow A$ and *D* is not a key. However if we split up *R* as *AD*, *BCD* we cannot preserve the dependency $ABC \rightarrow D$. So there is no BCNF decomposition.

4. (a) Candidate keys: *A*
   (b) *R* is in 2NF but not 3NF (because of the FD: $BC \rightarrow D$).
   (c) $BC \rightarrow D$ violates BCNF since *BC* does not contain a key. So we split up *R* as in: *BCD*, *ABC*.

5. (a) Candidate keys: *AB*, *BC*, *CD*, *AD*
   (b) R is in 3NF but not BCNF (because of the FD: C → A).
   (c) $C \rightarrow A$ and $D \rightarrow B$ both cause violations. So decompose into: *AC*, *BCD* but this does not preserve $AB \rightarrow C$ and $AB \rightarrow D$, and *BCD* is still not BCNF because $D \rightarrow B$. So we need to decompose further into: *AC*, *BD*, *CD*. However, when we attempt to revive the lost functioanl dependencies by adding *ABC* and *ABD*, we that these relations are not in BCNF form. Therefore, there is no BCNF decomposition.

Exercise 19.8 Consider the attribute set $R = ABCDEGH$ and the FD set $F = \{AB \rightarrow C,\ AC \rightarrow B,\ AD \rightarrow E,\ B \rightarrow D,\ BC \rightarrow A,\ E \rightarrow G\}$.

1. For each of the following attribute sets, do the following: (i) Compute the set of dependencies that hold over the set and write down a minimal cover. (ii) Name the strongest normal form that is not violated by the relation containing these attributes. (iii) Decompose it into a collection of BCNF relations if it is not in BCNF.

   (a) *ABC*, (b) *ABCD*, (c) *ABCEG*, (d) *DCEGH*, (e) *ACEH*

2. Which of the following decompositions of $R = ABCDEG$, with the same set of dependencies *F*, is (a) dependency-preserving? (b) lossless-join?
   (a) $\{AB, BC, ABDE, EG\}$
   (b) $\{ABC, ACDE, ADG\}$

Answer 19.8

1. (a) i. $R1 = ABC$: The FD's are $AB \rightarrow C$, $AC \rightarrow B$, $BC \rightarrow A$.

ii. This is already a minimal cover.

iii. This is in BCNF since AB, AC and BC are candidate keys for *R1*. (In fact, these are all the candidate keys for *R1*).

(b) i. *R2* $= ABCD$: The FD's are $AB \rightarrow C$, $AC \rightarrow B$, $B \rightarrow D$, $BC \rightarrow A$.

ii. This is a minimal cover already.

iii. The keys are: AB, AC, BC. *R2* is not in BCNF or even 2NF because of the FD, $B \rightarrow D$ (B is a proper subset of a key!) However, it is in 1NF. Decompose as in: ABC, BD. This is a BCNF decomposition.

(c) i. *R3* $= ABCEG$; The FDs are $AB \rightarrow C$, $AC \rightarrow B$, $BC \rightarrow A$, $E \rightarrow G$.

ii. This is in minimal cover already.

iii. The keys are: ABE, ACE, BCE. It is not even in 2NF since E is a proper subset of the keys and there is a FD $E \rightarrow G$. It is in 1NF . Decompose as in: ABE, ABC, EG. This is a BCNF decompostion.

(d) i. *R4* $= DCEGH$; The FD is $E \rightarrow G$.

ii. This is in minimal cover already.

iii. The key is $DCEH$; It is not in BCNF since in the FD $E \rightarrow G$, E is a subset of the key and is not in 2NF either. It is in 1 NF Decompose as in: $DCEH$, EG

(e) i. *R5* $= ACEH$; No FDs exist.

ii. This is a minimal cover.

iii. Key is $ACEH$ itself.

iv. It is in BCNF form.

2. (a) The decomposition. { AB, BC, ABDE, EG } is *not* lossless. To prove this consider the following instance of R:

$\{(a_1, b, c_1, d_1, e_1, g_1),\ (a_2, b, c_2, d_2, e_2, g_2)\}$

Because of the functional dependencies $BC \rightarrow A$ and $AB \rightarrow C$, $a_1 \neq a_2$ if and only if $c_1 \neq c_2$. It is easy to that the join AB $\bowtie$ BC contains 4 tuples:

$\{(a_1, b, c_1),\ (a_1, b, c_2),\ (a_2, b, c_1),\ (a_2, b, c_2)\}$

So the join of AB, BC, $ABDE$ and EG will contain at least 4 tuples, (actually it contains 8 tuples) so we have a lossy decomposition here.

This decomposition does not preserve the FD, $AB \rightarrow C$ (or $AC \rightarrow B$)

(b) The decomposition {ABC, ACDE, ADG } is lossless. Intuitively, this is because the join of ABC, $ACDE$ and ADG can be constructed in two steps; first construct the join of ABC and ACDE: this is lossless because their (attribute) intersection is AC which is a key for $ABCDE$ (in fact $ABCDEG$) so this is lossless. Now join this intermediate join with ADG. This is also lossless because the attribute intersection is is AD and $AD \rightarrow ADG$. So by the test mentioned in the text this step is also a lossless decomposition.

The projection of the FD's of R onto ABC gives us: $AB \rightarrow C$, $AC \rightarrow B$ and $BC \rightarrow A$. The projection of the FD's of R onto $ACDE$ gives us: $AD \rightarrow E$ and The projection of the FD's of R onto ADG gives us: $AD \rightarrow G$ (by transitivity) The closure of this set of dependencies does not contain $E \rightarrow G$ nor does it contain $B \rightarrow D$. So this decomposition is not dependency preserving.

Exercise 19.9 Let R be decomposed into R_1, R_2, ..., R_n. Let F be a set of FDs on R.

1. Define what it means for F to *be preserved* in the set of decomposed relations.
2. Describe a polynomial-time algorithm to test dependency-preservation.
3. Projecting the FDs stated over a set of attributes X onto a subset of attributes Y requires that we consider the closure of the FDs. Give an example where considering the closure is important in testing dependency-preservation, that is, considering just the given FDs gives incorrect results.

Answer 19.9

1. Let F_i denote the projection of F on R_i. F is *preserved* if the closure of the (union of) the F_i's equals F (note that F is always a superset of this closure.)
2. We shall describe an algorithm for testing dependency preservation which is polynomial in the cardinality of F. For each dependency X → Y ∈ F check if it is in F as follows: start with the set S (of attributes in) X. For each relation R_i, compute the closure of $S \cap R_i$ relative to F and project this closure to the attributes of R_i. If this results in additional attributes, add them to S. Do this repeatedly until there is no change to S.
3. There is an example in the text in Section 19.5.2.

Exercise 19.10 Suppose you are given a relation *R(A,B,C,D)*. For each of the following sets of FDs, assuming they are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) State whether or not the proposed decomposition of R into smaller relations is a good decomposition and briefly explain why or why not.

1. $B \rightarrow C$, $D \rightarrow A$; decompose into BC and AD.
2. $AB \rightarrow C$, $C \rightarrow A$, $C \rightarrow D$; decompose into ACD and BC.
3. $A \rightarrow BC$, $C \rightarrow AD$; decompose into ABC and AD.
4. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB and ACD.

5. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB, AD and CD.

Answer 19.10

1. Candidate key(s): BD. The decomposition into BC and AD is unsatisfactory because it is lossy (the join of BC and AD is the cartesian product which could be much bigger than $ABCD$)

2. Candidate key(s): AB, BC. The decomposition into ACD and BC is lossless since $ACD \cap BC$ (which is C) $\rightarrow ACD$. The projection of the FD's on ACD include C $\rightarrow$ D, C $\rightarrow$ A (so C is a key for ACD) and the projection of FD on BC produces no nontrivial dependencies. In particular this is a BCNF decomposition (check that R is not!). However, it is not dependency preserving since the dependency AB $\rightarrow$ C is not preserved. So to enforce preservation of this dependency (if we do not want to use a join) we need to add ABC which introduces redundancy. So implicitly there is some redundancy across relations (although none inside ACD and BC).

3. Candidate key(s): A, C. Since A and C are both candidate keys for R, it is already in BCNF. So from a normalization standpoint it makes no sense to decompose R. Further more, the decompose is not dependency-preserving since $C \rightarrow AD$ can no longer be enforced.

4. Candidate key(s): A. The projection of the dependencies on AB are: $A \rightarrow B$ and those on ACD are: $A \rightarrow C$ and $C \rightarrow D$ (rest follow from these). The scheme ACD is not even in 3NF, since C is not a superkey, and D is not part of a key. This is a lossless-join decomposition (since A is a key), but not dependency preserving, since $B \rightarrow C$ is not preserved.

5. Candidate key(s): A (just as before) This is a lossless BCNF decomposition (easy to check!) This is, however, not dependency preserving (B *consider* $\rightarrow C$). So it is not free of (implied) redundancy. This is not the best decomposition (the decomposition AB, BC, CD is better.)

Exercise 19.11 Consider a relation R that has three attributes ABC. It is decomposed into relations R_1 with attributes AB and R_2 with attributes BC.

1. State the definition of a lossless-join decomposition with respect to this example. Answer this question concisely by writing a relational algebra equation involving R, R_1, and R_2.

2. Suppose that $B \rightarrow C$. Is the decomposition of R into R_1 and R_2 lossless-join? Reconcile your answer with the observation that neither of the FDs $R_1 \cap R_2 \rightarrow R_1$ nor $R_1 \cap R_2 \rightarrow R_2$ hold, in light of the simple test offering a necessary and sufficient condition for lossless-join decomposition into two relations in Section 15.6.1.

3. If you are given the following instances of R_1 and R_2, what can you say about the instance of R from which these were obtained? Answer this question by listing tuples that are definitely in R and tuples that are possibly in R.

   Instance of $R_1 = \{(5,1), (6,1)\}$
   Instance of $R_2 = \{(1,8), (1,9)\}$

   Can you say that attribute B definitely *is* or *is not* a key for R?

Answer 19.11

1. The decomposition of R into R1 and R2 is lossless if and only if:

$$R1 \bowtie_{R1.B=R2.B} R2 \quad = \quad R$$

   Note that this is a statement about relation schemas, not some specific instances of them.

2. Answer Omitted.

3. All we can say is that the instance of R from which the given instances of R_1 and R_2 were obtained, must be a subset of the set of ABC tuples: {(5,1,8), (5,1,9), (6,1,8), (6,1,9)} which is also, at the same time, a superset of {(5,1,8), (6,1,9)} or a superset of {(5,1,9), (6,1,8)}. In particular, R contains at least two tuples but no more than 4. This also implies the attribute B is *not* a key for R (because R has at least 2 distinct tuples but each tuple in R has the same B value.)

Exercise 19.12 Suppose that we have the following four tuples in a relation S with three attributes ABC: (1,2,3), (4,2,3), (5,3,3), (5,3,4). Which of the following functional ($\rightarrow$) and multivalued ($\rightarrow\rightarrow$) dependencies can you infer does *not* hold over relation S?

1. $A \rightarrow B$
2. $A \rightarrow\rightarrow B$
3. $BC \rightarrow A$
4. $BC \rightarrow\rightarrow A$
5. $B \rightarrow C$
6. $B \rightarrow\rightarrow C$

Answer 19.12

1. (A $\rightarrow$ B) Cannot say anything.

2. (A $\to\to$ B) Cannot say anything.
3. (BC $\to$ A) does *not* hold. (Look at the tuples (1,2,3) and (4,2,3) the BC-values are the same but A values differ.)
4. (BC $\to\to$ A) Cannot say anything.
5. (B $\to$ C) does *not* hold. (Look at the tuples (5,3,3) and (5,3,4))
6. (B $\to\to$ C) Cannot say anything. The tuples (5,3,3) and (5,3,4) could shed some light if their A-values differed but that is not the case, here.

   In summary, we can conclude from the given information that BC $\to$ A and B $\to$ C do not hold.

Exercise 19.13 Consider a relation R with five attributes $ABCDE$.

1. For each of the following instances of R, state whether it violates (a) the FD $BC \to D$ and (b) the MVD $BC \to\to D$:
   (a) { } (i.e., empty relation)
   (b) {(a,2,3,4,5), (2,a,3,5,5)}
   (c) {(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)}
   (d) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5)}
   (e) {(a,2,3,4,5), (2,a,3,7,5), (a,2,3,4,6)}
   (f) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)}
   (g) {(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)}
2. If each instance for R listed above is legal, what can you say about the FD $A \to B$?

Answer 19.13

1. **Note**: The answer sometimes depends on the value of a. Unless otherwise mentioned, the answer applies to all values of a.

   (a) { } (i.e., empty relation):
   does not violate either dependency.

   (b) {(a,2,3,4,5), (2,a,3,5,5)}:
   If a = 2, then BC $\to$ D is violated (otherwise it is not).
   BC $\to\to$ D is not violated (for any value of a)

(c) {(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)}:
BC → D is violated if a = 2 (otherwise not).
If a = 2 then BC →→ D is violated (consider the tuples (2,a,3,5,5) and (a,2,3,4,6); if a equals 2 must also have (2,a,3,5,6))

(d) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5)}:
BC → D is violated (consider the first and the third tuples (a,2,3,4,5) and (a,2,3,6,5)).
BC →→ D is not violated.

(e) {(a,2,3,4,5), (2,a,3,7,5), (a,2,3,4,6)}:
If a = 2, then BC → D is violated (otherwise it is not).
If a = 2, then BC →→ D is violated (otherwise it is not). To prove this look at the last two tuples; there must also be a tuple (2,a,3,7,6) for BC →→ to hold.

(f) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)}:
BC → D does not hold. (Consider the first and the third tuple).
BC →→ C is violated. Consider the 1st and the 4th tuple. For this dependency to hold there should be a tuple (a,2,3,4,6).

(g) {(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)}:
BC → D does not hold. (Consider the first and the third tuple).
BC →→ C is not violated.

2. We can *not* say anything about the functional dependency A → B.

Exercise 19.14 JDs are motivated by the fact that sometimes a relation that cannot be decomposed into two smaller relations in a lossless-join manner can be so decomposed into three or more relations. An example is a relation with attributes *supplier*, *part*, and *project*, denoted *SPJ*, with no FDs or MVDs. The JD $\bowtie \{SP,\ PJ,\ JS\}$ holds.

From the JD, the set of relation schemes *SP, PJ,* and *JS* is a lossless-join decomposition of *SPJ.* Construct an instance of *SPJ* to illustrate that no two of these schemes suffice.

Answer 19.14
Consider the following instance of the schema SPJ:

$$SPJ = \{(s_1, p_1, j_1),\ (s_2, p_1, j_2),\ (s_1, p_2, j_2),\ (s_1, p_1, j_2)\}$$

Then

$$\begin{aligned} SP &= \{(s_1, p_1),\ (s_1, p_2),\ (s_2, p_1)\} \\ PJ &= \{(p_1, j_1),\ (p_1, j_2),\ (p_2, j_2)\} \\ JS &= \{(j_1, s_1),\ (j_2, s_1),\ (j_2, s_2)\} \end{aligned}$$

Let us compute all three 2-joins:

$$\begin{aligned} SP \bowtie PJ &= \{(s_1,p_1,j_1),(s_1,p_1,j_2),(s_2,p_1,j_1),(s_2,p_1,j_2),(s_1,p_2.j_2)\} \\ PJ \bowtie JS &= \{(s_1,p_1,j_1),(s_1,p_1,j_2),(s_1,p_2,j_2),(s_2,p_1,j_2),(s_2,p_2,j_2)\} \\ SP \bowtie JS &= \{(s_1,p_1,j_1),(s_1,p_1,j_2),(s_1,p_2,j_1),(s_1,p_2,j_2),(s_2,p_1,j_2)\} \end{aligned}$$

none of which equals SPJ. But, on the other hand, SPJ is the join of all three (if you put sufficiently many "constraints " you can always reconstruct the original relation SPJ by taking the join. Joining SP, PJ and JS amounts to putting all possible equality constraints. I am only giving you the intuition here; you need to work this out and check the details!) **QED.**

Exercise 19.15 Answer the following questions

1. Prove that the algorithm shown in Figure 19.4 correctly computes the attribute closure of the input attribute set X.
2. Describe a linear-time (in the size of the set of FDs, where the size of each FD is the number of attributes involved) algorithm for finding the attribute closure of a set of attributes with respect to a set of FDs. Prove that your algorithm correctly computes the attribute closure of the input attribute set.

Answer 19.15 The answer to each question is given below.

1. Proof Omitted.
2. Recall that the *attribute closure* of (attribute) X relative to a set of FD's Σ is the set of attributes A such that Σ satisfies $X \rightarrow A$.

```
// Initialize
$X^+$     := $X$;
FdSet := $\Sigma$;

do
{
      for each FD $Y \rightarrow Z$ in FdSet such that $X^+ \supseteq Y$
      {
          $X^+$ := $X^+$ union $Z$;
          Remove $Y \rightarrow Z$ from FdSet;
      }
} until ( $X^+$ does not change) ;
```

Let $n = \mid \Sigma \mid$ denote the cardinality of Σ. Then the loop repeats at most n times since for each iteration we either permanently remove a functional dependency from the set Σ, or stop all together. With the proper choice of data structures it can be show that this algorithm is linear in the size of Σ.

As for correctness, we claim that this algorithm is equivalent to the standard attribute closure algorithm. If we throw away a functional dependency $Y \rightarrow Z$ at a given step, then it must be the case that $X \rightarrow Y$ since $Y \in X^+$ at that step, therefore by transitivity $X \rightarrow Z$. Since Z was added to X^+ we no longer need the functional dependency $Y \rightarrow Z$ for finding the attribute closure of X, since it is implied by $X \rightarrow X^+$.

The rest of the proof of correctness follows from part 1 of this exercise.

Exercise 19.16 Let us say that an FD $X \rightarrow Y$ is *simple* if Y is a single attribute.

1. Replace the FD $AB \rightarrow CD$ by the smallest equivalent collection of simple FDs.
2. Prove that every FD $X \rightarrow Y$ in a set of FDs F can be replaced by a set of simple FDs such that F^+ is equal to the closure of the new set of FDs.

Answer 19.16

1. We claim $\{\ AB \rightarrow C,\ AB \rightarrow D\ \}$ is the smallest such set. First, this collection is equivalent to the single FD: $AB \rightarrow CD$ *i.e.* every database that satisfies the first also satisfies the second, and vice versa. Also no proper subset of this collection satisfies this property.
2. Replace each FD: $X_1X_2 \ldots X_m \rightarrow Y_1Y_2 \ldots Y_n$ by the collection $\{X_1 \ldots X_m \rightarrow Y_i \mid i \leq i \leq n\}$. By using the *decomposition* and *union* axioms, it is easy to show that we can go from one representation to the other both forwards and back. Note however, that this may not be the minimal such set.

Exercise 19.17 Prove that Armstrong's Axioms are sound and complete for FD inference. That is, show that repeated application of these axioms on a set F of FDs produces exactly the dependencies in F^+.

Answer 19.17 Proof Omitted.

Exercise 19.18 Consider a relation R with attributes $ABCDE$. Let the following FDs be given: $A \rightarrow BC$, $BC \rightarrow E$, and $E \rightarrow DA$. Similarly, let S be a relation with attributes $ABCDE$ and let the following FDs be given: $A \rightarrow BC$, $B \rightarrow E$, and $E \rightarrow DA$. (Only the second dependency differs from those that hold over R.) You do not know whether or which other (join) dependencies hold.

1. Is R in BCNF?
2. Is R in 4NF?
3. Is R in 5NF?
4. Is S in BCNF?
5. Is S in 4NF?
6. Is S in 5NF?

Answer 19.18

1. The schema R has keys A, E and BC. It follows that R is indeed in BCNF.
2. By Exercise 26 (part 1), it follows that R is also in 4NF (since the relation scheme has a single-attribute key).
3. R is in 5NF because the schema does not have any JD (besides those that are implied by the FD's of the schema; but these cannot violate the 5NF condition). Note that this alternative argument may be used in some of the other parts of this problem as well.
4. The schema S has keys A, B and E. It follows that S is indeed in BCNF.
5. By exercise 26 (part 1), it follows that S is also in 4NF (since the relation scheme has a single-attribute key).
6. By exercise 26 (part 2), it follows that S is also in 5NF (since each key is a single-attribute key.)

Exercise 19.19 Let R be a relation schema with a set F of FDs. Prove that the decomposition of R into R_1 and R_2 is lossless-join if and only if F^+ contains $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$.

Answer 19.19 For both directions (if and only-if) we use the notation

$C = R_1 \cap R_2, \quad X = R_1 - C, \quad Y = R_2 - C$, so that$R_1 = XC, \quad R_2 = CY$, and $R = XCY$.

(⇐): For this direction, assume we are given the dependency $C \rightarrow X$. (The other case $C \rightarrow Y$ is similar.)

So let r be an instance of schema R and let (x_1, c, y_1) and (x_2, c, y_2) be two tuples in r. The FD, $C \rightarrow X$ implies that $x_1 = x_2$. Thus, (x_1, c, y_2) is the same as (x_2, c, y_2) and (x_2, c, y_1) is the same as (x_1, c, y_1), so that both these tuples (x_1, c, y_2) and (x_2, c, y_1) are in r. Thus r satisfies the JD: $R = R_1 \bowtie R_2$. Since r is an arbitrary instance, we have proved that the decomposition is lossless.

(⇒): Now for the other direction, assume that neither $C \rightarrow X$ nor $C \rightarrow Y$ holds. We shall prove that the join is lossy by exhibiting a relation instance that violates the JD: $R_1 \bowtie R_2$. Actually we will prove a slightly more general result. Suppose we are given some set of FD's Σ, such that R has a lossless join w.r.t. Σ. This means that for any instance r satisfying Σ, we have

$$r = r_1 \bowtie r_2 \text{ where } r_1 = \pi_{R_1}(r),\ r_2 = \pi_{R_2}(r).$$

Then we prove that

$$\{C \rightarrow X,\ C \rightarrow Y\} \ \cap\ \Sigma^+ \neq \emptyset.$$

The proof is by contradiction. Assume that the intersection $\{\ C \rightarrow X, C \rightarrow Y\ \} \cap \Sigma^+$ is empty. Suppose r_1 is an instance of the schema that does not satisfy the FD: $C \ \rightarrow\ X$ and r_2 is an instance that does not satisfy the FD: $C \ \rightarrow\ Y$. Choose c such that there are tuples $(x_1, c, y_1),\ (x_2, c, y_2) \in r_1$ for which $x_1 \neq x_2$ and c' such that there are tuples $(x_1', c', y_1'),\ (x_2', c', y_2') \in r_2$ for which $y_1{}' \neq y_2{}'$.

Use selection to replace r_1 by $\pi_{R.C=c}(r_1)$ and r_2 by $\pi_{R.C=c'}(r_2)$. Since r_1 and r_2 are finite and the domain sets are infinite, we can assume without loss of generality (by modifying some of the values of the tuples in r_1 and r_2, if necessary) so that

$$\begin{array}{rcll} c & = & c' & \\ \pi_A(r_1) \ \cap\ \pi_A(r_2) & = & \emptyset & \text{for each attribute } A \in X \\ \pi_B(r_1) \ \cap\ \pi_B(r_2) & = & \emptyset & \text{for each attribute } B \in Y. \end{array}$$

Now consider the relation $r_1 \cup r_2$. This is an instance of the schema R that satisfies Σ. However, $(x_1, c, y_1') \notin r_1 \cup r_2$, so the instance $r_1 \cup r_2$ does not satisfy the JD: $R_1 \bowtie R_2$.

Exercise 19.20 Consider a scheme R with FDs F that is decomposed into schemes with attributes X and Y. Show that this is dependency-preserving if $F \ \subseteq\ (F_X \cup F_Y)^+$.

Answer 19.20 We need to show that $F \ \subseteq\ (F_X \cup F_Y)^+$ implies $F^+ \ =\ (F_X \cup F_Y)^+$. We can do this by showing the containments, $(F_X \cup F_Y)^+ \ \subseteq\ F^+$ and $F^+ \subseteq (F_X \cup F_Y)^+$, are both true. We make use the following two observations:

1. If $A \ \subseteq\ B$ are two sets of FD's then $A^+ \ \subseteq\ B^+$ and
2. $A^{++} = A^+$.

The includsion $(F_X \cup F_Y)^+ \ \subseteq\ F^+$ follows from observing that, by definition, $F_X \ \subseteq\ F^+$ and $F_Y \ \subseteq\ F^+$ so that $F_X \cup F_Y \ \subseteq\ F^+$ (now apply observations 1 and 2).

The other containment, $F^+ \subseteq (F_X \cup F_Y)^+$ follows from the hypothesis, $F \subseteq (F_X \cup F_Y)^+$ and observations 1 and 2.

Therefore, since both containments are true, $F^+ = (F_X \cup F_Y)^+$ also holds, which means the decomposition is dependency preserving.

Exercise 19.21 Prove that the optimization of the algorithm for lossless-join, dependency-preserving decomposition into 3NF relations (Section 19.6.2) is correct.

Answer 19.21 Answer Omitted.

Exercise 19.22 Prove that the 3NF synthesis algorithm produces a lossless-join decomposition of the relation containing all the original attributes.

Answer 19.22
Proof: Let R denote the set of all attributes. N is a minimal cover for the set of all FD's satsified by the schema and K some key for the schema. We will show that the decomposition $\{XA | X \rightarrow A \in N\} \quad \{K\}$, where K is any key gives a lossless join 3NF decomposition. First, note that the subschema K is in 3NF because any FD that holds over it will have its right hand side (attribute) contained in a key (namely, K).

The proof that the decomposition is lossless is a little complicated notationally but the basic idea is this: Enumerate the set of subschema XA in the decomposition as $R_1, R_2, \ldots R_m$. Let r be an instance and let t_i and a be tuples in r. We need to show the "joins" of all these tuples are also in r. A formal proof of this will proceed by induction and is based on using the tuple a and the fact that K is a key dependency that follows from the FD's $X \rightarrow A$ to "connect" the other tuples and force each tuple in the join to lie in r.

Exercise 19.23 Prove that an MVD $X \rightarrow\rightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie \{XY,\ X(R-Y)\}$.

Answer 19.23 Write $Z = R - Y$. Thus, $R = YXZ$. $X \rightarrow\rightarrow Y$ says that if $(y_1, x, z_1), (y_2, x, z_2) \in R$ then $(y_1, x, z_2), (y_2, x, z_1)$ also $\in R$. But this is precisely the same as saying $R = \bowtie \{\ XY,\ X(R-Y)\ \}$.

Exercise 19.24 Prove that, if R has only one key, it is in BCNF if and only if it is in 3NF.

Answer 19.24 Let F (F^+) denote the (closure of the) set of functional dependencies satisfied by the schema R which is assumed to be in 3NF. We need to show that for each nontrivial dependency $X \rightarrow A$ in F^+, X is a superkey. To this end, consider such a dependency. If X is *not* a superkey, the 3NF property guarantees that the attribute A is part of a key. Since all keys are simple by assumption, we have that A is a key. This last fact together with the dependency $X \rightarrow A$ implies that X is a superkey (this follows, from the transitivity axiom) which is a contradiction.

Exercise 19.25 Prove that, if R is in 3NF and every key is simple, then R is in BCNF.

Answer 19.25 Since every key is simple, then we know that for any FD that satisfies $X \rightarrow A$, where A is part of some key implies that A is a key. By the definition of an FD, if X is known, then A is known. This means that if X is known, we know a key for the relation, so X must be a superkey. This satisfies all of the properties of BCNF.

Exercise 19.26 Prove these statements:

1. If a relation scheme is in BCNF and at least one of its keys consists of a single attribute, it is also in 4NF.
2. If a relation scheme is in 3NF and each key has a single attribute, it is also in 5NF.

Answer 19.26 The answer to each question is given below.

1. Proof Omitted.
2. Proof Omitted.

Exercise 19.27 Give an algorithm for testing whether a relation scheme is in BCNF. The algorithm should be polynomial in the size of the set of given FDs. (The *size* is the sum over all FDs of the number of attributes that appear in the FD.) Is there a polynomial algorithm for testing whether a relation scheme is in 3NF?

Answer 19.27 Let $|F|$ denote the size of the representation of the schema *i.e.,* set of all the FD's of the schema. Also, let $|f|$ denote the number of FD's in the schema. By exercise 19.15 we know that for each attribute in the schema, we can compute the attribute closure of its left hand side in time $O(|F|)$.

The algorithm to test if R is in BCNF consists of computing the attribute closure of the left hand side of each FD. If one of them doesn not equal U where U is the set of all attributes, then R is not in BCNF. Otherwise conclude that R is in BCNF.

Clearly the worst case complexity of this algorithm is $O(|f| \cdot |F|)$. Since $|f|$ is bounded by $|F|$ this yields a polynomial time algorithm in $|F|$.

On the other hand, *a priori*, there is no polynomial time algorithm for testing for 3NF. This is because to test whether or not a given FD violates 3NF we may need to check if the right hand side is prime *i.e.,* is a subset of some key of the schema. But identifying (all) the keys of the schema involves checking all subsets of U, and there are $2^{|U|}$ many of them. This last *prime attribute* problem is known to be NP-complete and the 3NF problem is clearly as hard (in fact polynomially reducible to the other) and, hence is also NP-complete.

Exercise 19.28 Give an algorithm for testing whether a relation scheme is in BCNF. The algorithm should be polynomial in the size of the set of given FDs. (The 'size' is the sum over all FDs of the number of attributes that appear in the FD.) Is there a polynomial algorithm for testing whether a relation scheme is in 3NF?

Answer 19.28 Incorrect question listed in the textbook. Please see solution to Exercise 19.27.

Exercise 19.29 Prove that the algorithm for decomposing a relation schema with a set of FDs into a collection of BCNF relation schemas as described in Section 19.6.1 is correct (i.e., it produces a collection of BCNF relations, and is lossless-join) and terminates.

Answer 19.29 First, we will repeat the algorithm so as to keep consistent notation:

1. Let $X \subset R$, A be a single atribute in R and $X \rightarrow A$ be a FD that causes a violation of BCNF. Decompose into $R - A$ and XA.
2. If either $R - A$ or XA is not in BCNF, decompose them further by a recursive application of this algorithm.

Proving the correctness of the algorithm is divided into 3 parts:

- **Proof that every Decomposition is Lossless:**
  For any decomposition of a relation R into $R - A$ and XA that the algorithm takes, it is trivially loseless by Thoerem 3 of this chapter. First, we claim that $(R - A) \bigcap (XA) = X$ since: $X \subset R$ by construction, and A is not in X (else it would be a trivially functional dependency and not violate BCNF, which is a contradiction). The given functional dependency $X \rightarrow A$ then implies $X \rightarrow XA$ by the Union Rule, therefore $(R - A) \bigcap (XA) \rightarrow XA$ and by Thoerem 3, this decomposition is loseless. Note however, that this decomposition may not be dependency preserving.

- **Proof that Algorithm Terminates:**
  Every decomposition of a Relation R that the algorithm performs produces relations $R - A$ and XA with strictly fewer attributes then R. $R - A$ has strictly fewer attributes than R since by construction A is not null, and since the functional dependency violates BCNF also by construction, A must be contained in R, else the functional dependency would not be applicable to R. Further, XA has strictly fewer attributes than R since by construction $X \subset R$ and $XA \neq R$. This is clear since if we assume that $XA = R$, then we can conclude that X is a superkey because $XA \rightarrow R$ trivially and $X \rightarrow A$, so that $X \rightarrow R$ from Transivitivty. This

would contradict the assumption that the functional dependency violates BCNF, leaving us to conclude that $XA \neq R$.

If we let n denote the number of attributes in the original relation R, then there are at most $(2^n - 1)$ decompositions the algorithm will perform before it terminates. Once a relation contains just a single attribute, it is in BCNF and cannot be decomposed further since there are no non-trivial functional dependencies we can apply to it that would violate BCNF.

- **Proof that every Relation in Final Set is in BCNF:**
  As discussed in the previous part of the problem, in the worst case the algorithm decomposes R into a set of n unique single attribute relations where n is the number of attributes in the original relation R. As also discussed above, each relation is clearly in BCNF. The decomposition process may, and in most cases should, terminate before we are down to all single attribute relations but irregardless, the algorithm will only stop when all subsets of R are in BCNF.