Solutions

Q1 \ XY\rightarrow Z, XZ\rightarrow Y \ and \ YZ\rightarrow X \ all \ holds. \ There \ are \ many \ non-trivial \ (and \ related) \ FDs. \ Thanks \ for \ the \ hardwork.

Q2.1
It \ holds:
CD\rightarrow BD \ (F3, \ augmentation)
CD\rightarrow A \ (Above + F2, \ transitivity)

Q2.2
BC\rightarrow A \ does \ not \ hold. \ Consider \ the \ following \ counter \ example \ with \ 2 \ tuples:
a1 \ b1 \ c1 \ d1
a2 \ b1 \ c1 \ d2

Q2.3
It \ holds:
AD\rightarrow A \ (reflexitivity)
A\rightarrow B \ (F1 + F3, \ transitivity)
AD\rightarrow B \ (above \ two, \ transitivity)
Q3.1
{A, D}

Q3.2
{A, B, C, D}

Q3.3
A->D and B->CD

Q3.4
Notice A and B must appear in the candidate key since they do not appear on the right hand side of any dependencies. So the only candidate key is {A, B}.

Q4.1
It is neither loss-less nor dependency-preserving. AG->E is not preserved.

Q4.2
It is dependency-preserving, but not lossless.

Q4.3
It is lossless: First we can join {ABCE} and {AEG} lossless-ly, since {AE} is candidate key for {AGE}. Then, {ABCEG} and {BD} can be again joined lossless-ly since {B} is a candidate key for {BD}.

It is also dependency-preserving.

Q5.1
The candidate key is {AC} and {BC}.

Q5.2
R is not in BCNF and not in 3NF. Notice for dependency B->D, B is not superkey, and D does not appear in the candidate key.

Q5.3
They are two ways to decompose. Notice BC->A does not violate BCNF ({BC} is a superkey), so we can start by either A->B or B->D.
Case1: If we start from A->B, we decompose R to {AB, ACD}. Notice that now A->D holds, so {ACD} is not in BCNF, thus it needs to be further decomposed to {AC, AD}. It is not dependency-preserving since both BC->A and B->D are lost.

Case2: If we start from B->D, we decompose R to {BD, ABC}. Then since {ABC} is not in BCNF, it is further split to {AB, AC}. It is not dependency-preserving since BC->A is lost.

Q5.4

Many of you use 3NF synthesis to construct a dependency-preserving 3NF decomposition.

If using the instructions in page 627:

Case1: Starting from {AB, AC, AD}, which is a lossless 3NF decomposition. Notice that it is not dependency-preserving (BC->A and B->D). So we add another two relations {ABC, BD}. The result is {ABC, BD, AB, AC, AD} (You can eliminate either AB or AC since it is included in {ABC}).

Case2: Starting from {BD, ABC}, which is a lossless 3NF decomposition. Notice that it’s already dependency-preserving, so it’s the final answer.