Decomposition

Steps:

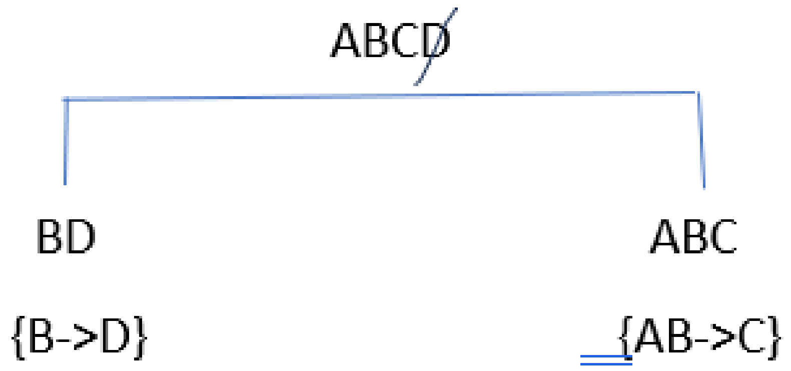
i) Identify the Candidate Keys(CK)
ii) Identify the problematic FDs
iii) Begin with the original list of attributes
iv) For each of the Problematic FDs
   a. Compute the closure of the left side of that FD
   b. Grab all the attributes from the closure and create one table with those attributes
   c. Cut off the extra attributes of the Closure from the original list of attributes.
v) Check if all the decomposed tables are at least in the desired normal form.
   - If not, then for every decomposed table that are not at least in the desired normal form, repeat this process.
vi) Check if the decomposition is preserving the original list of functional dependencies
vii) Check if the decomposition is lossless.

Example-1:

Convert to 2NF: R(ABCD) FD={AB->C, B->D}. Show the decomposition is FD preserving and lossless.

1) CK = AB

2) Problematic FDs: B->D (not in 2NF)
3) Original attribute list: ABCD.
4) B^{+}={BD}, cutting off D as it is an extra attribute to B's closure.



5) No more problematic FDs. So we grab the surviving attributes (ABC) and create a second table with them

<u>Check if R1(BD) in 2NF</u>:

Yes, because we have only two attributes for this table. So, it is already in BCNF and by default it means that its in 2NF as well

<u>Check if R2(ABC) in 2NF</u>:

Yes, because the CK for R2(ABC) is AB. The FD for R2(ABC) has all of the CK on the left side (not any part of it), hence its in 2NF.

<u>Functional Dependency check:</u>

R1(BD) FD1={B->D} and R2(ABC) FD2={AB->C}

From the original FD list:
1) AB->C can be generated from R2(ABC)
2) B->C can be generated from R1(BD).

So, this is a FD preserving decomposition.

<u>Lossless property check:</u>

The decomposed tables: R1(BD), R2(ABC)

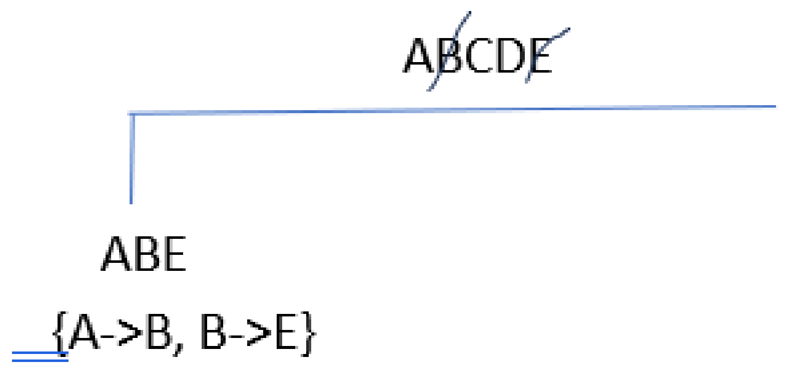
i) attribute(R1) U attribute(R2) = attribute(R)
ii) common attribute = R1 ∩ R2 = B. B is the Super Key of R1(BD) as B^+={BD}

So, this decomposition is lossless.

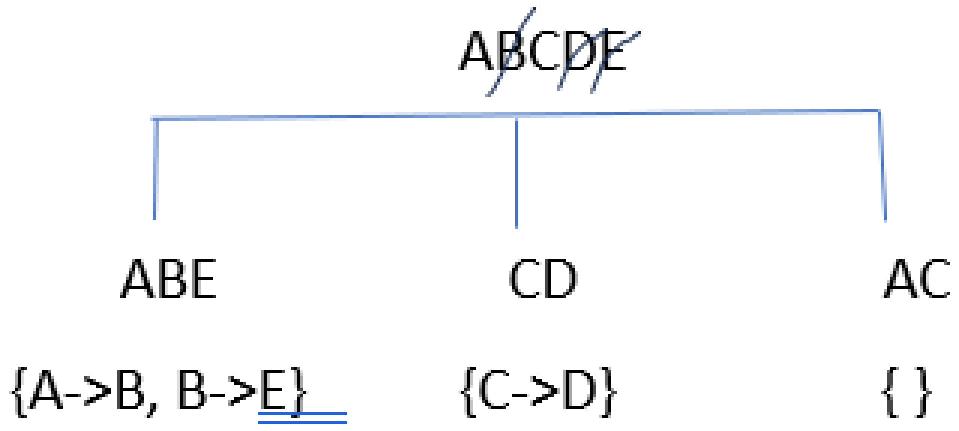
Example-2:

Convert to 2NF: R(ABCDE) FD={A->B, B -> E, C ->D}. Show the decomposition is FD preserving and lossless.

1) CK = AC
2) Problematic FDs: A->B, C->D (not in 2NF)
3) Original attribute list: ABCDE.
4) A^+={ABE}, cutting off B,E as they are extra attribute to A's closure.



5) C^+={CD}, cutting off D as it is an extra attribute to C’s closure.



6) No more problematic FDs. So we grab the surviving attributes (AC) and create a third table with them

<u>Check if R(ABE) in 2NF</u>:

Yes, because the CK for R(ABE) is A. The FD for R(ABC) has all of the CK on the left side (not any part of it), hence its in 2NF.

<u>Check if R(CD) in 2NF</u>:

Yes, because we have only two attributes for this table. So, it is already in BCNF and by default it means that its in 2NF as well

Check if R(AC) in 2NF:

Yes, because we have only two attributes for this table. So, it is already in BCNF and by default it means that its in 2NF as well

Functional Dependency check:

R1(ABE) FD1={A->B, B->E}, R2(CD) FD2={C->D}, and R3(AC) FD3={ }

From the original FD list:
1) A->B, B->E can be generated from R1(ABE)
2) C->D can be generated from R2(CD).

So, this is a FD preserving decomposition.

Lossless property check:

The decomposed tables: R1(ABE), R2(CD), R3(AC)

i) attribute(R1) U attribute(R2) U attribute(R3) = attribute(R)
ii) common attribute = R1 ∩ R3 = A, A is the super key of R3(AC)
iii) common attribute = R2 ∩ (R1 U R3) = C, C is the super key of R2(CD)

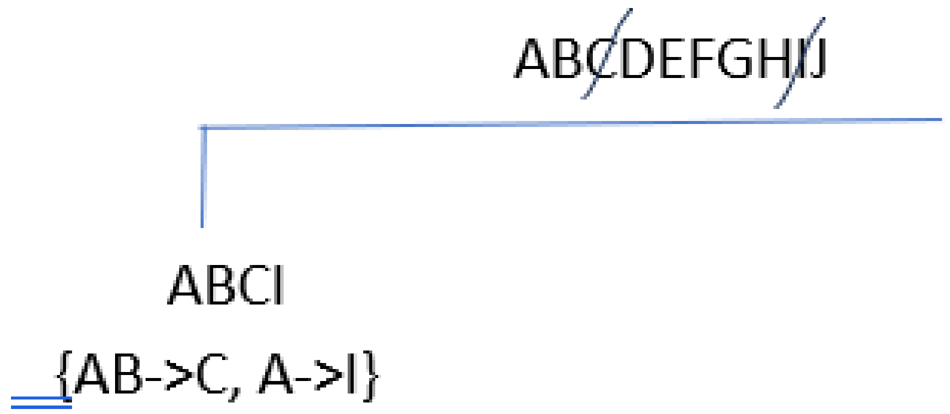
So, this decomposition is lossless.

Example-3:

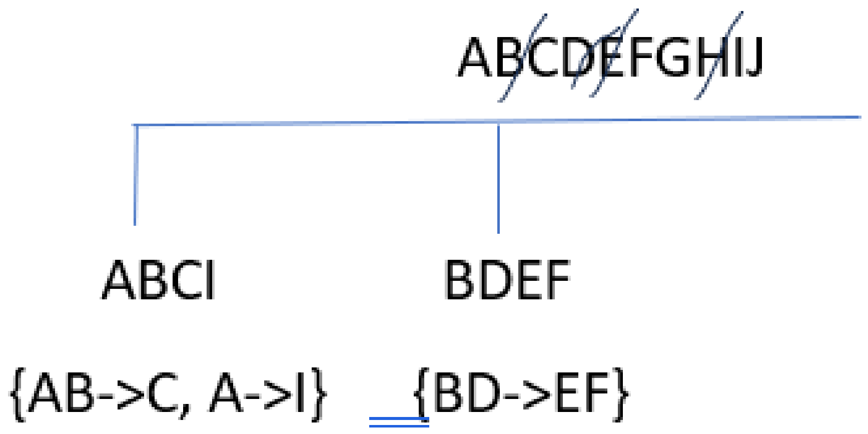
Convert to 2NF: R(ABCDEFGHIJ)

FD={AB->C, BD -> EF, AD->GH, A->I, H ->J}. Show the decomposition is FD preserving and lossless.

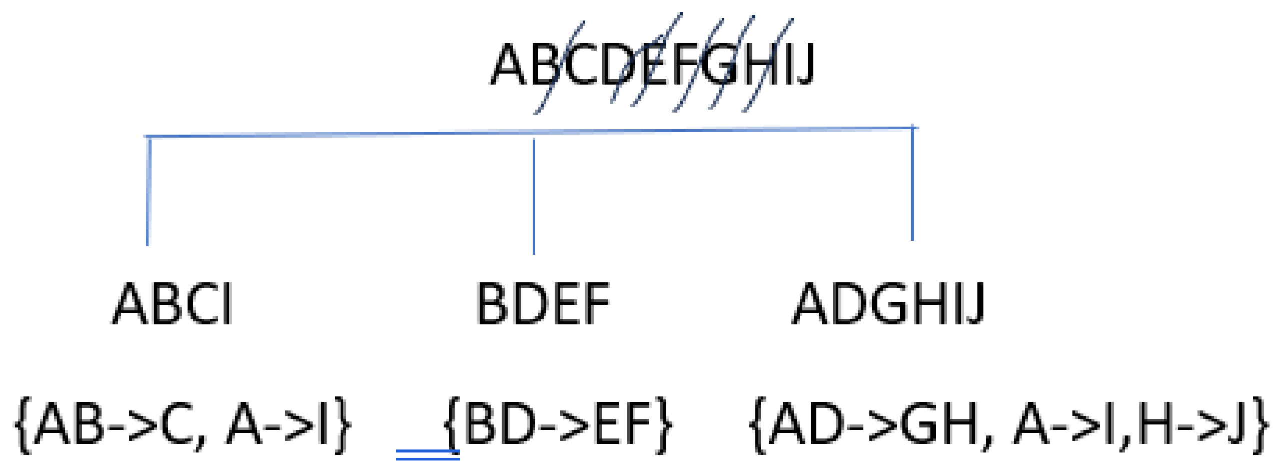
1) CK = ABD
2) Problematic FDs: AB->C, BD -> EF, AD->GH, A->I (not in 2NF)
3) Original attribute list: ABCDEFGHIJ.
4) AB$^+$={ABCI}, cutting off C,I as they are extra attribute to AB's closure.



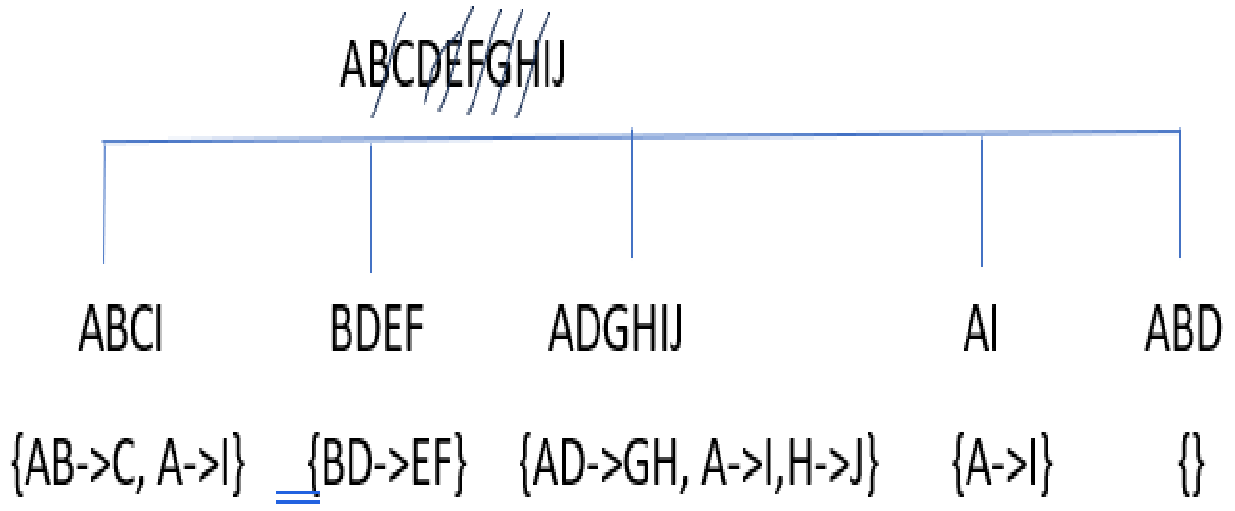
5) BD$^+$={BDEF}, cutting off E,F as they are extra attribute to BD's closure.



6) AD$^+$={ADGHIJ}, cutting off G,H,I,J as they are extra attribute to AD's closure.



7) AI$^+$={AI}, cutting off I as its an extra attribute to AI's closure.

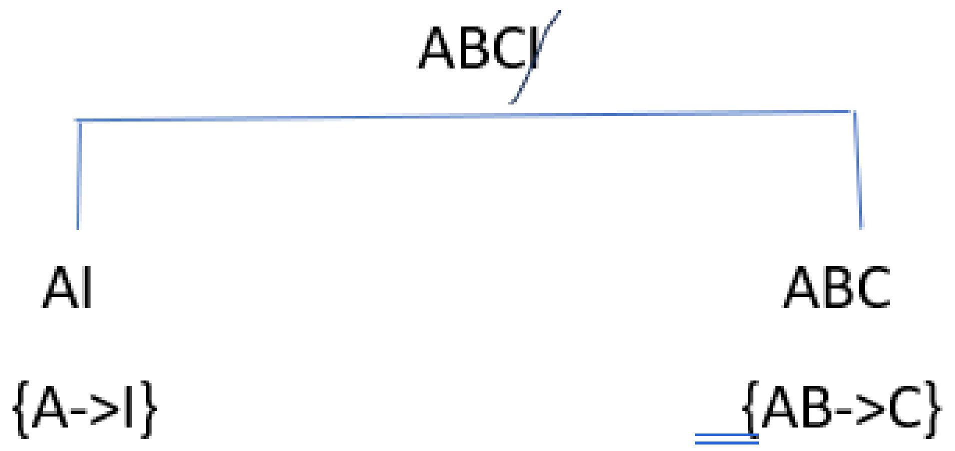


8) No more problematic FDs. So we grab the surviving attributes (ABD) and create another table with them.

<u>Check if R1(ABCI) in 2NF</u>:

NO, the CK for this table is AB. The second FD for this table is A->I, has (a) part of the CK on the left side and (b) the right side is from NPA. So, it violated 2NF. We need to further decompose R1(ABCI).

- CK is AB
- Problematic FD: {A->I}
- Original attribute list for this table: ABCI
- AI^+={AI}, cutting off I as its an extra attribute to AI's closure.



We have already created a table with AI. So, no need to create a duplicate table. So, we only create a new table with (ABC).

Check if R(ABC) in 2NF:

Yes, because the CK for R(ABC) is AB. The FD for R(ABC) has all of the CK on the left side (not any part of it), hence its in 2NF.

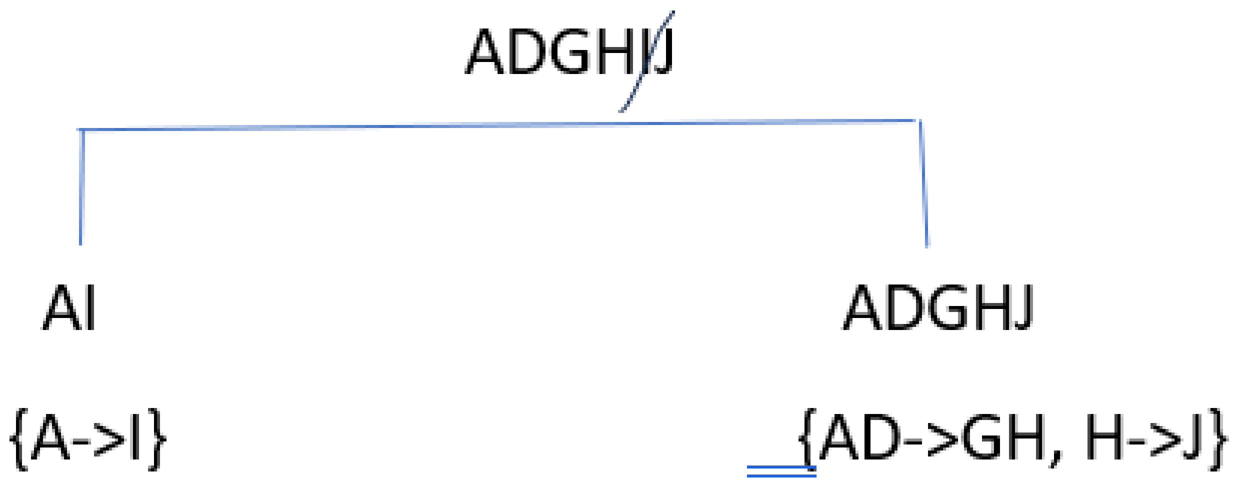
Check if R(BDEF) in 2NF:

Yes, because the CK for R(BDEF) is BD. The FD for R(BDEF) has all of the CK on the left side (not any part of it), hence its in 2NF.

Check if R(ADGHIJ) in 2NF:

NO, because the CK for R(ADGHIJ) is AD. The two FDs for R(ADGHIJ) AD->GH and H->J is in 2NF. But A->I is not. So we further decompose this table.

- CK is AD
- Problematic FD: {A->I}
- Original attribute list for this table: ADGHIJ
- AI^+={AI}, cutting off I as its an extra attribute to AI's closure.



- We have already created a table with AI. So, no need to create a duplicate table. So, we only create a new table with (ADGHJ).

Check if R(ABD) in 2NF:

Yes, because R(ABC) has an empty FD list. So, its in BCNF meaning its by default in 2NF.

So, all the decomposed tables are in 2NF. The decomposed tables:

R1(ABC), R2(BDEF), R3(ADGHJ), R4(AI), R5(ABD)

<u>Functional Dependency check:</u>

R1(ABC) FD1={AB->C}, R2(BDEF) FD2={BD->EF}, R3(ADGHJ) FD3={AD->GH, H->J} R4(AI) FD4={A->I} R5(ABD) FD5={ }

From the original FD list:
1) AB->C can be generated from R1(ABC)
2) BD -> EF can be generated from R2(BDEF)
3) AD->GH, H->J can be generated from R3(ADGHJ)
4) A -> I can be generated from R4(AI)

So, this is a FD preserving decomposition.

<u>Lossless property check:</u>

The decomposed tables: R1(ABC), R2(BDEF), R3(ADGHJ), R4(AI), R5(ABD)

i) attribute(R1) U attribute(R2) U attribute(R3) U attribute(R4) U attribute(R5) = attribute(R)
ii) common attribute = R2 ∩ R5 = BD, BD is the super key of R2(BDEF)
iii) common attribute = R3 ∩ (R2 U R5) = AD, AD is the super key of R3(ADGHJ)

iv) common attribute = R4 ∩ (R3 U R2 U R5) = A, A is the super key of R4(AI)
v) common attribute = R1 ∩ (R4 U R3 U R2 U R5) = AB, AB is the super key of R1(ABC)

So, this decomposition is lossless.

PRACTICE PROBLEMS:

Decompose into 2NF:

1) R(ABCDEF) {AB->C, C->D, B->E, B->F}
2) R(ABCDEFG) {AB->C, E->G, A->E, B->F}

Decompose into 3NF:

3) R(ABCDEF) {AB->C, C->D, D->E, B->F}
4) R(ABCD) {A->B, B->C, C->D}