Canonical cover
Lossless join
dependency preserving joins

Work out

Practice Problems on Functional Dependencies

PPD

Find Prime and Non Prime Attributes using Functional Dependencies:

1. R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F}
2. R(ABCDEF) having FDs {AB → C, C → DE, E → F, C → B}

PA : ABC
Non PA = DEF

3. R(ABCDEFGHIJ) having FDs {AB → C, A → DE, B → F, F → GH, D → IJ}
4. R(ABDLPT) having FDs {B → PT, A → D, T → L}
5. R(ABCDEFGH) having FDs {E → G, AB → C, AC → B, AD → E,B → D, BC → A}
6. R(ABCDE) having FDs {A → BC, CD → E, B → D, E → A}
7. R(ABCDEH) having FDs {A → B, BC → D, E → C, D → A}

- **Prime Attributes –** Attribute set that belongs to any candidate key are called Prime Attributes
  - It is union of all the candidate key attribute: {CK1 ∪ CK2 ∪ CK3 ∪ ……}
  - If Prime attribute determined by other attribute set, then more than one candidate key is possible.
  - For example, If A is Candidate Key, and X→A, then, X is also Candidate Key .
- **Non Prime Attribute –** Attribute set does not belongs to any candidate key are called Non Prime Attributes

Source: http://www.edugrabs.com/prime-and-non-prime-attributes/

R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F}

Candidate keys ?

$A^+ = A$

$AB^+ = ABCDEF = R \therefore AB$ is CK

~~AC^+ =~~ $ACDEFB = R \therefore AC$ is CK

$AD^+ = ABCDEF = R \therefore AD$ is CK

$AE^+ = ABCDEF = R \therefore AE$ is CK

~~$AF^+ = A$~~ $F\ BCDE = R \therefore AF$ is CK

Prime attributes : A B C D E F

Non Prime attribute : nil

Algorithms for Functional Dependencies

Lossless Join Decomposition

Dependency Preservation

ALGORITHMS FOR FUNCTIONAL DEPENDENCIES

Canonical Cover

Sets of functional dependencies may have redundant dependencies that can be inferred from the others

For example: $A \to C$ is redundant in: $\{A \to B, \quad B \to C, A \to C\}$

Parts of a functional dependency may be redundant

E.g.: on RHS: $\{A \to B, \quad B \to C, \quad A \to CD\}$ can be simplified to

$\{A \to B, \quad B \to C, \quad A \to D\}$

$A \to CD$

In the forward: (1) $A \to CD \Rightarrow A \to C$ and $A \to D$ (2) $A \to B, \quad B \to C \Rightarrow A \to C$

$A \to D$

In the reverse: (1) $A \to B, \quad B \to C \Rightarrow A \to C$ (2) $A \to C, A \to D \Rightarrow A \to CD$

E.g.: on LHS: $\{A \to B, \quad B \to C, \quad AC \to D\}$ can be simplified to

$\{A \to B, \quad B \to C, \quad A \to D\}$

In the forward: (1) $A \to B, \quad B \to C \Rightarrow A \to C \Rightarrow A \to AC$ (2) $A \to AC, AC \to D \Rightarrow A \to D$

In the reverse: $A \to D \Rightarrow AC \to D$

no-redundant

Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Canonical Cover

A **canonical cover** for F is a set of dependencies F_c such that

- F logically implies all dependencies in F_c, and
- F_c logically implies all dependencies in F, and
- No functional dependency in F_c contains an extraneous attribute, and
- Each left side of functional dependency in F_c is unique

To compute a canonical cover for F:

repeat

1 Use the union rule to replace any dependencies in F
$\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

2 Find a functional dependency $\alpha \rightarrow \beta$ with an extraneous attribute either in α or in β
/* Note: test for extraneous attributes done using F_c, not F*/

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until F does not change

Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

- *Minimal Sets of Functional Dependencies*
- *Irreducible Set of Functional Dependencies*

Canonical Cover: RHS

F   Fc

$\{A \to B,\ B \to C,\ A \to CD\} \to \{A \to B,\ B \to C,\ A \to D\}$

$(1)\ A \to CD \to A \to C \text{ and } A \to D\ (2)\ A \to B,\ B \to C \to A \to C$

$A+ = ABCD$

$\{A \to B,\ B \to C,\ A \to D\} \to \{A \to B,\ B \to C,\ A \to CD\}$

$A \to B,\ B \to C \to A \to C$

$A \to C,\ A \to D \to A \to CD$

$A+ = ABCD$

F = Fc

Canonical Cover: LHS

$\{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\} \rightarrow \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$

$A \rightarrow B,\ B \rightarrow C \rightarrow A \rightarrow C \rightarrow A \rightarrow AC$

$A \rightarrow AC,\ AC \rightarrow D \rightarrow A \rightarrow D$

$A+ = ABCD$

$\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\} \rightarrow \{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\}$

$A \rightarrow D \rightarrow AC \rightarrow D$

$AC+ = ABCD$

LHS $\alpha \rightarrow$ RHS β

Extraneous Attributes

Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.

- Attribute A is **extraneous** in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
- Attribute A is **extraneous** in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.

Note: Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one

Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$

- B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- A+ = AC in $\{A \rightarrow C, AB \rightarrow C\}$

Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$

- C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C
- $AB+ = ABCD$ *in* $\{A \rightarrow C, AB \rightarrow D\}$

Testing if an Attribute is Extraneous

Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

To test if attribute $A \in \alpha$ is extraneous in α

1. Compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. Check that $(\{\alpha\} - A)^+$ contains β; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β

3. Compute α^+ using only the dependencies in $F' =$ LHS +

   $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,

1. Check that α^+ contains *A*; if it does, *A* is extraneous in β

Computing a Canonical Cover

$\alpha_1 \to \beta_1$, $\alpha \to \beta_2$ $\cup$

Extraneous

Redundant

$R = (A, B, C)$ $F = \{A \to BC$, $B \to C$, $A \to B$, $AB \to C\}$

Combine $A \to BC$ and $A \to B$ into $A \to BC$

Set is now $\{A \to BC, B \to C, AB \to C\}$

A is extraneous in $AB \to C$

- Check if the result of deleting A from $AB \to C$ is implied by the other dependencies
  - Yes: in fact, $B \to C$ is already present!
- Set is now $\{A \to BC, B \to C\}$

C is extraneous in $A \to BC$

- Check if $A \to C$ is logically implied by $A \to B$ and the other dependencies
  - Yes: using transitivity on $A \to B$ and $B \to C$.
    - Can use attribute closure of A in more complex cases

$A \to B$, $B \to C$ $\Rightarrow$ $A \to C$

The canonical cover is:

$A \to B$
$B \to C$

minimal

$4 \Rightarrow 2$

Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:

1. AB → CD, BC → D

- Step 1:Use the union rule to replace any dependencies in *F*

$\alpha 1 \rightarrow \beta 1$ and $\alpha 1 \rightarrow \beta 2$ with $\alpha 1 \rightarrow \beta 1\ \beta 2$

So we can write AB → C, AB → D, BC → D

Step 2: find the redundancy step by step:

(AB)+ ={A,B,C,D} // Now check for AB → C

(AB)+ ={A,B,D} without considering this FD :AB → C

(A)+ ={A}

(B)+ ={B}

So AB → C is essential not redundancy.[left side multiple attribute is also essential]

(AB)+ ={A,B,C,D} // Now check for AB → D

(AB)+ ={A,B,C,D} without considering this FD :AB → D [we got the same closure]

So AB → D is redundancy.[SO delete this one]

(BC)+ ={B,C,D} // Now check for BC → D

(BC)+ ={B,C} without considering this FD :BC → D

SO this FD :BC → D is essential.

So canonical cover set of the above set of FD is AB → C, BC → D

Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:

2. $ABCD \to E$, $E \to D$, $AC \to D$, $A \to B$

- Step 1:Use the union rule to replace any dependencies in F

$\alpha 1 \to \beta 1$ and $\alpha 1 \to \beta 2$ with $\alpha 1 \to \beta 1\ \beta 2$

We have nothing like this.

Step 2: find the redundancy step by step:

Now check for $ABCD \to E$

$(ABCD)^+ = \{A,B,C,D,E\}$

$(ABCD)^+ = \{A,B,C,D\}$ without considering this FD : $ABCD \to E$

$(A)^+ = \{A,B\}$

$(B)^+ = \{B\}$

$(C)^+ = \{C\}$

$(D)^+ = \{D\}$

So $ABCD \to E$ is essential not redundancy.

Now check for $E \to D$

$(E)^+ = \{E,D\}$

$(E)^+ = \{E\}$ without considering this FD: $E \to D$,

So $E \to D$ is essential.

Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:

$ABCD \to E$, $E \to D$, $AC \to D$, $A \to B$

Now check for $AC \to D$,

$(AC)+ = \{A,C,D\}$ //USING $AC \to D$

$= \{A,B,C,D\}$ //USING $A \to B$

$= \{A,B,C,D,E\}$ //USING $ABCD \to E$

$(AC)+ = \{A,B,C\}$ without considering this FD : $AC \to D$

$(A)+=\{A,B\}$

$(C)+=\{C\}$

SO this FD : $AC \to D$ is essential.[LEFT HAND ATTRIBUTE IS ALSO ESSENTIAL]

Now check for $A \to B$

$(A)+=\{A,B\}$

$(A)+=\{A\}$ without considering this FD: $A \to B$

SO $A \to B$ this FD is essential

So canonical cover set of the above set of FD is $ABCD \to E$, $E \to D$, $AC \to D$, $A \to B$

Equivalence of Sets of Functional Dependencies

Let F & G are two functional dependency sets.

- These two sets F & G are equivalent if $F^+ = G^+$
- Equivalence means that every functional dependency in F can be inferred from G, and every functional dependency in G an be inferred from F

F and G are equal only if

- F covers G: Means that all functional dependency of G are logically numbers of functional dependency set F⇒F⊇G.
- G covers F: Means that all functional dependency of F are logically members of functional dependency set G⇒G⊇F

Condition	CASES			
F Covers G	True	True	False	False
G Covers F	True	False	True	False
Result	F=G	F⊃G	G⊃F	No Comparison

F

Practice Problems on Functional Dependencies

Check the Equivalence of a Pair of Sets of Functional Dependencies:

1. Consider the two sets F and G with their FDs as below :
   1. F : A → C, AC → D, E → AD, E → H
   2. G: A → CD, E → AH
2. Consider the two sets P and Q with their FDs as below :
   1. P : A → B, AB → C, D → ACE
   2. Q : A → BC, D → AE

Source: http://www.edugrabs.com/equivalence-of-sets-of-functional-dependencies-example/

Check the Equivalence of a Pair of Sets of Functional Dependencies:

1. Consider the two sets F and G with their FDs as below :

1. F : A → C, AC → D, E → AD, E → H
2. G: A → CD, E → AH

- Step 1 : Find out the closure of all lefthand attributes of F functional dependency(FD) using the FD of G and vice versa.

F : A → C, AC → D, E → AD, E → H	**G: A → CD, E → AH**
(A)+ ={A,C,D} //USING A → CD ,OF FD G	(A)+ ={A,C} //USING A → C, OF FD F ={A,C,D} //USING AC → D, OF FD F
(AC)+ ={A,C,D} //USING A → CD, OF FD G	(E)+={E,A,D,H} \\ USING E → AD, E → H ={E,A,C,D,H} \\ USING A → C
(E)+={A,E,H} // USING E → AH , OF FD G ={A,C,D,E,H} // USING A → CD	IF YOU WANT YOU CAN FIND FOR (AC)+ ={A,C,D} //USING AC → D OF F
F⊆G	G ⊆F

- F⊆G and G ⊆F
- So F=G
- So two sets of functional dependency is equivalent.

Check the Equivalence of a Pair of Sets of Functional Dependencies:

Q2. Consider the two sets F and G with their FDs as below :

1. P : A → B, AB → C, D → ACE
2. Q : A → BC, D → AE

- Step 1 : Find out the clousure of all lefthand attributes of P functional dependency(FD) using the FD of Q and vice versa.

P : A → B, AB → C, D → ACE	**Q : A → BC, D → AE**
(A)+ ={A,B,C} //USING A → BC // Q FD	(A)+ ={A,B} //USING A → B //FD OF P ={A,B,C} //USING AB → C, //FD OF P
(AB)+ ={A,B,C} //USING A → BC //Q FD	(D)+={D,A,C,E} //USING D → ACE //FD OF P ={A,B,C,D,E} //USING A → B //FD OF P
(D)+={D,A,E} // USING D → AE OF Q FD ={A,B,C,D,E} // USING A → BC,	IF YOU WANT YOU CAN FIND FOR (AB)+ ={A,B,C} //USING A → B, AB → C, //FD OF P

- P⊆Q and Q⊆P
- So P=Q
- So two sets of functional dependency is equivalent.

LOSSLESS JOIN DECOMPOSITION

Lossless-join Decomposition

For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R $\quad r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$

A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:

$R_1 \cap R_2 \rightarrow R_1$

$R_1 \cap R_2 \rightarrow R_2$

The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

To Identify whether a decomposition is lossy or lossless, it must satisfy the following conditions :

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \Phi$ *and*
- $R_1 \cap R_2 \rightarrow R_1$ *or* $R_1 \cap R_2 \rightarrow R_2$

Example

Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**

Having dependencies: **S# → Sname**, **S# → City**, **(S#, P#) → Qty**

Decompose as: **Supplier(S#, Sname, City, Qty)**: **Parts(P#, Qty)**

Take Natural Join to reconstruct: **Supplier** ⋈ **Parts**

R

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

R1

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

R2

P#	Qty
301	20
500	50
20	10
400	40
301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
5	**Nick**	**NY**	**20**	**10**
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10
2	**Steve**	**Boston**	**301**	**10**

We get extra tuples! **Join is Lossy!**

Common attribute **Qty** is not a superkey in **Supplier** or in **Parts**

Does not preserve **(S#,P#)** → **Qty**

Source: http://www.edugrabs.com/lossy-join-decomposition/

Example

Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**

Having dependencies: **S# → Sname**, **S# → City**, **(S#, P#) → Qty**

Decompose as: **Supplier(S#, Sname, City)**: **Parts(S#, P#, Qty)**

Take Natural Join to reconstruct: **Supplier** ⋈ **Parts**

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City
3	Smith	London
5	Nick	NY
2	Steve	Boston
5	Nick	NY
5	Nick	NY

S#	P#	Qty
3	301	20
5	500	50
2	20	10
5	400	40
5	301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

We get back the original relation. **Join is Lossless.**

Common attribute **S#** is a superkey in **Supplier**

Preserves all dependencies

Source: http://www.edugrabs.com/desirable-properties-of-decomposition/#lossless

Example

$R = (A, B, C)$
$F = \{A \rightarrow B, B \rightarrow C\}$

Can be decomposed in two different ways

$R_1 = (A, B), \quad R_2 = (B, C)$

Lossless-join decomposition:

$R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$

Dependency preserving

Ex2:

$R_1 = (A, B), \quad R_2 = (A, C)$

Lossless-join decomposition:

$R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$

Not dependency preserving
(cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Practice Problems on Lossless Join

Check if the decomposition of R into D is lossless:

1. R(ABC): F = {$A \rightarrow B$, $A \rightarrow C$}. D = R_1(AB), R_2(BC)
2. R(ABCDEF): F = {$A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $E \rightarrow F$}. D = R_1(AB), R_2(BCD), R_3(DEF)
3. R(ABCDEF): F = {$A \rightarrow B$, $C \rightarrow DE$, $AC \rightarrow F$}. D = R_1(BE), R_2(ACDEF)
4. R(ABCDEG): F = {$AB \rightarrow C$, $AC \rightarrow B$, $AD \rightarrow E$, $B \rightarrow D$, $BC \rightarrow A$, $E \rightarrow G$}
   1. D1 = R_1(AB), R_2(BC), R_3(ABDE), R_4(EG)
   2. D2 = R_1(ABC), R_2(ACDE), R_3(ADG)
5. R(ABCDEFGHIJ): F = {$AB \rightarrow C$, $B \rightarrow F$, $D \rightarrow IJ$, $A \rightarrow DE$, $F \rightarrow GH$}
   1. D1 = R_1(ABC), R_2(ADE), R_3(BF), R_4(FGH),R_5(DIJ)
   2. D2 = R_1(ABCDE), R_2(BFGH), R_3(DIJ)
   3. D3 = R_1(ABCD), R_2(DE), R_3(BF), R_4(FGH),R_5(DIJ)

Source: http://www.edugrabs.com/questions-on-lossless-join/

PPD

Algorithms for Functional Dependencies

Lossless Join Decomposition

Dependency Preservation

DEPENDENCY PRESERVATION

Dependency Preservation

Let F_i be the set of dependencies F^+ that include only attributes in R_i

A decomposition is **dependency preserving**, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$

If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Let R be the original relational schema having FD set F. Let R_1 and R_2 having FD set F_1 and F_2 respectively, are the decomposed sub-relations of R. The decomposition of R is said to be preserving if

- $F_1 \cup F_2 \equiv F$ *{Decomposition Preserving Dependency}*
- *If* $F_1 \cup F_2 \subset F$ *{Decomposition NOT Preserving Dependency} and*
- $F_1 \cup F_2 \supset F$ *{this is not possible}*

Testing for Dependency Preservation

To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into R_1, R_2, …, R_n we apply the following test (with attribute closure done with respect to F)

result = α
while (changes to *result*) do
for each R_i in the decomposition $t = (result \cap R_i)^+ \cap R_i$ $result = result \cup t$

If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.

We apply the test ~~on all~~ dependencies in F to check if a decomposition is dependency preserving

This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example

Test for dependency preserving.

R(ABCDEF):

F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E} ← check

D = {ABCD, BF, DE}

On projections:

given:

ABCD (R1)	BF (R2)	DE (R3)
A → BCD BC → AD	B → F	D → E

Need to check for: A→BCD, **A→EF**, BC→AD, **BC→E**, **BC→F**, B→F, D→E

using

(BC)+/F1 = ABCD. (ABCD)+/F2 = ABCDF. **(ABCDF)+/F3 = ABCDEF**. Preserves **BC→E**, **BC→F**

(A)+/F1 = ABCD. (ABCD)+/F2 = ABCDF. **(ABCDF)+/F3 = ABCDEF**. Preserves **A→EF**

1. BC+ using F1 = ABCD
2. (ABCD)+ using F2 = ABCDF
3. (ABCDF)+ using F3 ABCDEF

BC → E

→ F

Example

R1 R2 R3

R(ABCDEF): F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}. D = {ABCD, BF, DE}

On projections:

After

ABCD (R1)	BF (R2)	DE (R3)
$A \to B$, $A \to C$, $A \to D$, $BC \to A$, $BC \to D$	$B \to F$	$D \to E$

Infer reverse FD's:

- B+/F = BF: B → A cannot be inferred
- C+/F = C: C → A cannot be inferred
- D+/F = DE: D → A and D → BC cannot be inferred
- A+/F = ABCDEF: A → BC can be inferred, but it is equal to A → B and A → C
- F+/F = F: F → B cannot be inferred
- E+/F = E: E → D cannot be inferred

This is new method

But use last slide method

Need to check for: A→BCD, **A→EF**, BC→AD, **BC→E**, **BC→F**, B→F, D→E

- **(BC)+/F = ABCDEF**. Preserves **BC→E**, **BC→F**
- **(A)+/F = ABCDEF**. Preserves **A→EF**

Practice Problems on Dependency Preservation

Check whether the decomposition of R into D is preserving dependency:

1. R(ABCD): F = {A → B, B → C, C → D, D → A}. D = {AB, BC, CD}
2. R(ABCDEF): F = {AB → CD, C → D, D → E, E → F}. D = {AB, CDE, EF}
3. R(ABCDEG): F = {AB → C, AC → B, BC → A, AD → E, B → D, E → G}. D = {ABC, ACDE, ADG}
4. R(ABCD): F = {A → B, B → C, C → D, D → B}. D = {AB, BC, BD}
5. R(ABCDE): F = {A → BC, CD → E, B → D, E → A}. D = {ABCE, BD}

Source: http://www.edugrabs.com/question-on-dependency-preserving-decomposition/