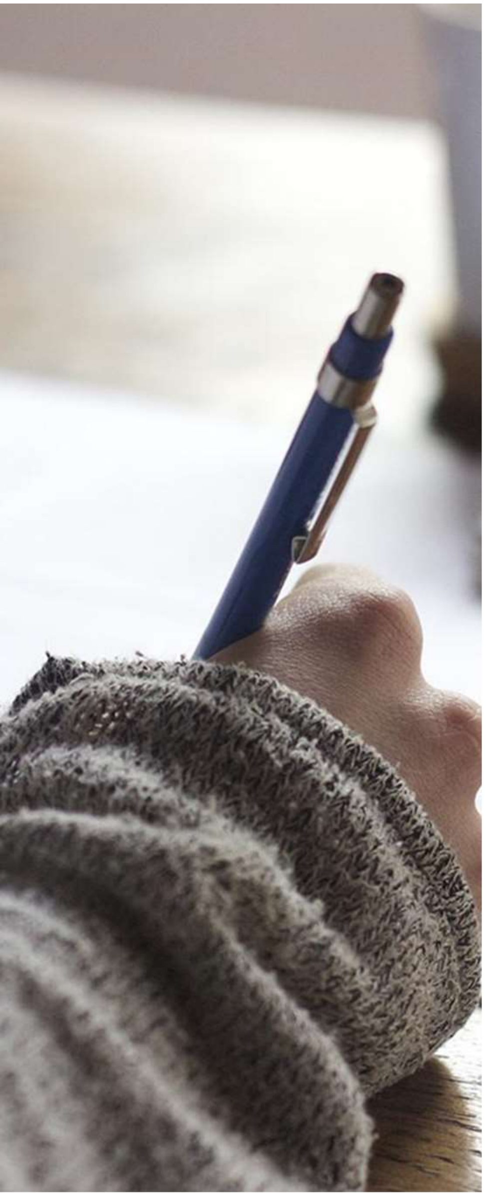
Decompositions

Decompositions

- Decomposition of a relation is done when a relation in relational model is not in appropriate normal form.
- Relation R is decomposed into two or more relations if decomposition is **lossless join** as well as **dependency preserving**.

Decompositions

- If R(A, B, C) satisfies $A \rightarrow B$
  - We can project it on A, B and A,C ***without losing information***
  - **Lossless** decomposition vs. **Lossy** decomposition
- If we decompose a relation R(A, B, C) into relations
  - $R1 = \pi_{AB}(R)$ and $R2 = \pi_{AC}(R)$
  - $\pi_{AB}(R)$ is the projection of R on AB
  - $\bowtie$ is the natural join operator
- Decomposition is **lossy** if $R \subset R1 \bowtie R2$
- Decomposition is **lossless** if $R = R1 \bowtie R2$



Decompositions

$$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$$

$$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) \qquad R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$$

R_1 = the *projection* of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = the *projection* of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Properties of Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

We need a decomposition to be "correct"

I.e. it is a **Lossless decomposition**

Lossy Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Need to avoid "bad" decompositions

What's wrong here?

Lossy Decomposition

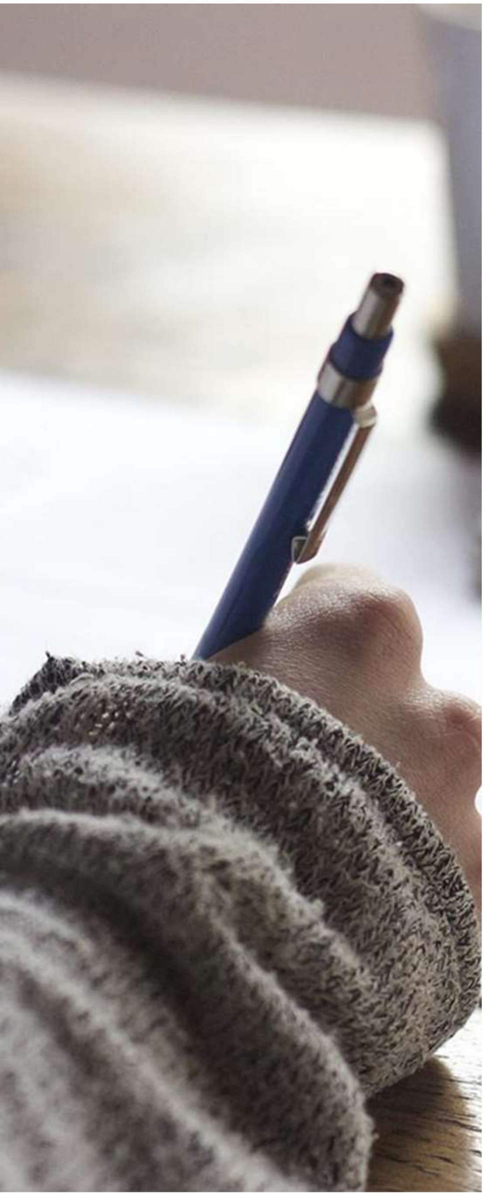
Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

$\bowtie$

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera
OneClick	19.9 9	Camera
[illegible]	24.9	[illegible]



Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

A decomposition R to (R1, R2) is **lossless** if R = R1 ⋈ R2

To check for lossless join decomposition using FD set, following conditions must hold:

1- Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

$$Att(R1) \cup Att(R2) = Att(R)$$

2- Intersection of Attributes of R1 and R2 must not be NULL.

$$Att(R1) \cap Att(R2) \neq \Phi$$

3- Common attribute must be a key for at least one relation (R1 or R2).

$$Att(R1) \cap Att(R2) \rightarrow Att(R1)$$

or

$$Att(R1) \cap Att(R2) \rightarrow Att(R2)$$

Example

A relation R (A, B, C, D) with FD set { A -> BC} is decomposed into R1(ABC) and R2(AD)

Is lossless join decomposition?

First condition holds **true** as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).

Second condition holds **true** as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ

Third condition holds **true** as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

Dependency Preserving Decomposition

If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.

For Example, A relation R (A, B, C, D) with FD set { A -> BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A -> BC is a part of R1(ABC).

Question

Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is

A. dependency preserving and lossless join

B. lossless join but not dependency preserving

C. dependency preserving but not lossless join

D. not dependency preserving and not lossless join

Answer

For **lossless join** decomposition, these three conditions must hold true:

Att(R1) U Att(R2) = ABCD = Att(R)

Att(R1) ∩ Att(R2) = Φ, which violates the condition of lossless join decomposition. Hence the decomposition is not lossless.

For **dependency preserving** decomposition,

A -> B can be ensured in R1(AB) and C -> D can be ensured in R2(CD). Hence it is dependency preserving decomposition.

So, the correct option is C.

Watch this video for database index

Indexing in DBMSs: B-Trees and B^+-Trees

https://www.youtube.com/watch?v=aZjYr87r1b8