Redundancy and Anomaly

- **Redundancy**: having multiple copies of same data in the database.
  - This problem arises when a database is not normalized
  - It leads to anomalies
- **Anomaly**: inconsistencies that can arise due to data changes in a database with insertion, deletion, and update
  - These problems occur in poorly planned, un-normalised databases where all the data is stored in one table (a flat-file database)

  There can be three kinds of anomalies
  - *Insertions Anomaly*
  - *Deletion Anomaly*
  - *Update Anomaly*

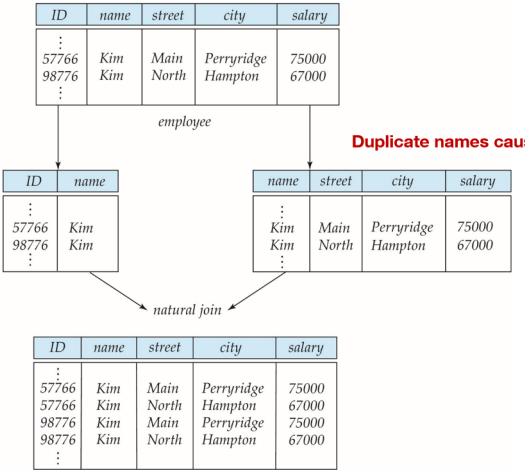
Redundancy and Anomaly (3)

- We have observed the following:
  - **Redundancy ⇒ Anomaly**
  - Relations *instructor* and *department* is better than *instructor_with_department*
- What causes redundancy?
  - **Dependency ⇒ Redundancy**
  - *dept_name* uniquely decides *building* and *budget*. A department cannot have two different budget or building. So *building* and *budget* **depends on** *dept_name*
- How to remove, or at least minimize, redundancy?
  - Decompose (partition) the relation into smaller relations
  - *instructor_with_department* can be decomposed into *instructor* and *department*
  - **Good Decomposition ⇒ Minimization of Dependency**
- Is every decomposition good?
  - No. It needs to preserve information, honour the dependencies, be efficient etc.
  - Various schemes of normalization ensure good decomposition
  - **Normalization ⇒ Good Decomposition**

Decomposition (2)

- Not all decompositions are good
- Suppose we decompose
  employee(ID, name, street, city, salary) into
  employee1 (ID, name)
  employee2 (name, street, city, salary)
- Note that if *name* can be duplicate, then *employee*2 is a weak entity set and cannot exist without an identifying relationship
- Consequently, this decomposition cannot preserve the information
- The next slide shows how we lose information – we cannot reconstruct the *original employee* relation – and so, this is a **lossy decomposition**.

Decomposition (3): Lossy Decomposition



Decomposition (5): Lossless-Join Decomposition

- **Lossless Join Decomposition** is a decomposition of a relation R into relations R_1, R_2 such that if we perform natural join of two smaller relations it will return the original relation

$$R_1 \cup R_2 = R, R_1 \cap R_2 \neq \phi$$

$$\forall r \in R, r_1 = \sqcap_{R_1}(r), r_2 = \sqcap_{R_2}(r)$$

$$r_1 \bowtie r_2 = r$$

- This is effective in removing redundancy from databases while preserving the original data
- In other words by lossless decomposition it becomes feasible to reconstruct the relation R from decomposed tables R_1 and R_2 by using Joins

First Normal Form (1NF)

- A domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **First Normal Form (INF)** if
  - the domains of all attributes of R are *atomic*
  - the value of each attribute contains only a *single value* from that domain

multivalued are not allowed

- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - *We assume all relations are in first normal form*

Functional Dependencies

- Constraints on the set of legal relations
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a *key*

Functional Dependencies (2)

- Let R be a relation schema
  $\alpha \subseteq R$ and $\beta \subseteq R$
- The functional dependency or FD
  $\alpha \rightarrow \beta$
  holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A, B)$ with the following instance of r.

A	B
1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold. So we cannot have tuples like (2, 4), or (3, 5), or (4, 7) added to the current instance.

Functional Dependencies (3)

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$ and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

  i.e. its minimal
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  inst_dept(ID, name, salary, dept_name, building, budget)
- We expect these functional dependencies to hold:

  dept_name $\rightarrow$ *building*

  dept_name $\rightarrow$ *budget*

  ID $\rightarrow$ *budget*

  but would not expect the following to hold:

  dept_name $\rightarrow$ *salary*

Functional Dependencies (4)

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation r is legal under a set F of functional dependencies, we say that r **satisfies** F
  - specify constraints on the set of legal relations
    - We say that F **holds on** R if all legal relations on R satisfy the set of functional dependencies F
- **Note**: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances
  - For example, a specific instance of instructor may, by chance, satisfy
    $$name \rightarrow ID$$
  - In such cases we do not say that F holds on R

Functional Dependencies (5)

- A functional dependency is trivial if it is satisfied by all instances of a relation
  - Example:
    - $ID, name \rightarrow ID$
    - $name \rightarrow name$
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.

Functional Dependencies: Armstrong's Axioms

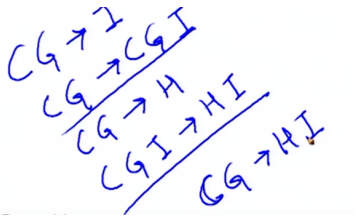
- Given a set of Functional Dependencies F, we can infer new dependencies by the **Armstrong's Axioms**:
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These axioms can be repeatedly applied to generate new FDs and added to F
- A new FD obtained by applying the axioms is said to the **logically implied** by F
- The process of generations of FDs terminate after finite number of steps and we call it the **Closure Set** F^+ for FDs F. This is the set of **all** FDs logically implied by F
- Clearly, $F \subseteq F^+$
- These axioms are
  - **Sound** (generate only functional dependencies that actually hold), and
  - **Complete** (eventually generate all functional dependencies that hold)
- Prove the axioms from definitions of FDs
- Prove the soundness and completeness of the axioms

Functional Dependencies (2): Closure of a Set FDs

- $F = \{A \rightarrow B, B \rightarrow C\}$
- $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Functional Dependencies (3): Closure of a Set FDs

- $R = (A, B, C, G, H, I)$
  $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$



- Some members of F^+
  - $A \rightarrow H$
    - ▷ by transitivity from $A \rightarrow B$ and $B \rightarrow H$
  - $AG \rightarrow I$
    - ▷ by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - ▷ by augmenting $CG \rightarrow I$ with CG to infer $CG \rightarrow CGI$, and augmenting $CG \rightarrow H$ with I to infer $CGI \rightarrow HI$, and then transitivity

Functional Dependencies (4): Closure of a Set FDs: Computing F^+

- To compute the closure of a set of functional dependencies F:

  $F^+ \leftarrow F$

  repeat

    for each functional dependency f in F^+

      apply reflexivity and augmentation rules on f

      add the resulting functional dependencies to F^+

    for each pair of functional dependencies f_1 and f_2 in F^+

      if f_1 and f_2 can be combined using transitivity

        then add the resulting functional dependency to F^+

  until F^+ does not change any further

- **Note**: We shall see an alternative procedure for this task later

Functional Dependencies (5): Armstrong's Axioms: Derived Rules

- Additional Derived Rules:
  - **Union**: if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition**: if $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity**: if $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds
- The above rules can be inferred from basic Armstrong's axioms (and hence are not included in the basic set). They can be proven independently too
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- Prove the Rules from:
  - Basic Axioms
  - The definitions of FDs

Functional Dependencies (6): Closure of Attribute Sets

- Given a set of attributes α, define the **closure** of α **under** F (denoted by α^+) as the set of attributes that are functionally determined by α under F
- Algorithm to compute α^+, the closure of α under F

```
result ← α
while (changes to result) do
        for each β → γ in F do
                begin
                        if β ⊆ result then result ← result ∪ γ
                end
```

Functional Dependencies (7): Closure of Attribute Sets: Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B,\ A \rightarrow C,\ CG \rightarrow H,\ CG \rightarrow I,\ B \rightarrow H\}$
- $(AG)^+$
  a) result = AG
  b) result = ABCG ($A \rightarrow C$ and $A \rightarrow B$)
  c) result = ABCGH ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  d) result = ABCGHI ($CG \rightarrow I$ and $CG \subseteq AGBCH$)
- Is AG a candidate key?
  a) Is AG a super key?
     i) Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  b) Is any subset of AG a superkey?
     i) Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     ii) Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Functional Dependencies (7): Closure of Attribute Sets: Use

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

BCNF: Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of FDs if for all FDs in F^+ of the form

  $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R

- Example schema *not in* BCNF:

  instr_dept (ID, name, salary, dept_name, building, budget)

- because the non-trivial dependency *dept_name* $\rightarrow$ *building, budget* holds on *instr_dept*, but *dept_name* is not a superkey

BCNF (2): Decomposition

- If in schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, we decompose R into:
  - $\alpha \cup \beta$
  - $(R - (\beta - \alpha))$
- In our example,
  - $\alpha =$ *dept_name*
  - $\beta =$ *building*, *budget*
  - *dept_name* $\rightarrow$ *building*, *budget*

  inst_dept is replaced by
  - $(\alpha \cup \beta) =$ (*dept_name*, *building*, *budget*)
    - *dept_name* $\rightarrow$ *building*, *budget*
  - $(R - (\beta - \alpha))$ =(*ID*, *name*, *salary*, *dept_name*)
    - *ID* $\rightarrow$ *name*, *salary*, *dept_name*

BCNF (3): Lossless Join

- If we decompose a relation R into relations R_1 and R_2:
  - Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
  - Decomposition is lossless if $R_1 \bowtie R_2 = R$
- To check for lossless join decomposition using FD set, following must hold:
  - Union of Attributes of R_1 and R_2 must be equal to attribute of R

$$R_1 \cup R_2 = R$$

  - Intersection of Attributes of R_1 and R_2 must not be NULL

$$R_1 \cap R_2 \neq \Phi$$

  - Common attribute must be a key for at least one relation (R_1 or R_2)

SUPER

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2$$

- Prove that BCNF ensures Lossless Join

BCNF (4): Dependency Preservation

- Constraints, including FDs, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- It is not always possible to achieve both BCNF and dependency preservation. Consider:
  - $R = CSZ$, $F = \{CS \rightarrow Z, Z \rightarrow C\}$
  - Key $= CS$
  - $CS \rightarrow Z$ satisfies BCNF, but $Z \rightarrow C$ violates
  - Decompose as: $R_1 = ZC, R_2 = CSZ - (C - Z) = SZ$
  - $R_1 \cup R_2 = CSZ = R$, $R_1 \cap R_2 = Z \neq \Phi$, and $R_1 \cap R_2 = Z \rightarrow ZC = R_1$. So it has *lossless join*
  - However, we cannot check $CS \rightarrow Z$ without doing a join. Hence it is not *dependency preserving*
- We consider a weaker normal form, known as **Third Normal Form (3NF)**

3NF: Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:
  $$\alpha \rightarrow \beta \in F^+$$
  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R
    (Nore: Each attribute may be in a different candidate key)

bcnf

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

Extraneous Attributes

- Consider a set F of FDs and the FD $\alpha \rightarrow \beta$ 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 FDs $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ 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 $A \rightarrow C$ (that is, 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 $\{A \rightarrow C, AB \rightarrow D\}$

Extraneous Attributes (2): Tests

- 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
  a) Compute $(\{\alpha\} - A)^+$ using the dependencies in F
  b) 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
  a) Compute α^+ using only the dependencies in $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  b) Check that α^+ contains A; if it does, A is extraneous in β

Equivalence of Sets of Functional Dependencies

- Let F & G are two functional dependency sets.
  - These two sets F & G are equivalent if $F^+ = G^+$. That is: $(F^+ = G^+) \Leftrightarrow (F^+ \Rightarrow G \text{ and } G^+ \Rightarrow F)$
  - 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 members of functional dependency set $F \Rightarrow F^+ \supseteq G$.
  - G covers F: Means that all functional dependency of F are logically members of functional dependency set $G \Rightarrow G^+ \supseteq F$.

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

Canonical Cover

- Sets of FDs may have redundant dependencies that can be inferred from the others
- Can we have some kind of "optimal" or "minimal" set of FDs wto work with?
- A **Canonical Cover** for F is a set of dependencies F_c such that ALL the following properties are satisfied:
  - $F^+ = F_c^+$. Or,
    - F logically implies all dependencies in F_c
    - F_c logically implies all dependencies in F
  - No functional dependency in F_c contains an extraneous attribute
  - Each left side of functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in such that $\alpha_1 \rightarrow \alpha_2$
- Intuitively, a **Canonical cover** of F is a **minimal** set of FDs
  - Equivalent to F
  - Having no redundant FDs
  - No redundant parts of FDs

more than one covers possible

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

Canonical Cover (5)

- To compute a canonical cover for F:

  repeat

  Use the union rule to replace any dependencies in F

  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$

  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

Practice Problems on Functional Dependencies (4)

- **Find Prime and Non Prime Attributes using Functional Dependencies:**
  a) $R(ABCDEF)$ having FDs $\{AB \rightarrow C, C \rightarrow D, D \rightarrow E, F \rightarrow B, E \rightarrow F\}$
  b) $R(ABCDEF)$ having FDs $\{AB \rightarrow C, C \rightarrow DE, E \rightarrow F, C \rightarrow B\}$
  c) $R(ABCDEFGHIJ)$ having FDs $\{AB \rightarrow C, A \rightarrow DE, B \rightarrow F, F \rightarrow GH, D \rightarrow IJ\}$
  d) $R(ABDLPT)$ having FDs $\{B \rightarrow PT, A \rightarrow D, T \rightarrow L\}$
  e) $R(ABCDEFGH)$ having FDs $\{E \rightarrow G, AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A\}$
  f) $R(ABCDE)$ having FDs $\{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$
  g) $R(ABCDEH)$ having FDs $\{A \rightarrow B, BC \rightarrow D, E \rightarrow C, D \rightarrow A\}$

- **Prime Attributes**: Attribute set that belongs to any candidate key are called Prime Attributes
  - It is union of all the candidate key attribute: $\{CK_1 \cup CK_2 \cup CK_3 \cup \cdots\}$
  - If Prime attribute determined by other attribute set, then more than one candidate key is possible.
  - For example, If A is Candidate Key, and $X \rightarrow A$, then, X is also Candidate Key.
- **Non Prime Attribute**: Attribute set does not belong to any candidate key are called Non Prime Attributes

Lossless Join Decomposition (4): Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways
- $R_1 = (A, B), R_2 = (B, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), R_2 = (A, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i
  - A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$$

  - 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}

Dependency Preservation (2): Testing

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $D = \{R_1, R_2, \ldots, R_n\}$ we apply the following test (with attribute closure done with respect to F)
- The **restriction** of F^+ to R_i is the set of all functional dependencies in F^+ that include only attributes of R_i.
  - compute F^+;
    for each schema R_i in D **do**
      begin
        F_i = the restriction of F^+ to R_i;
      end
    $F' = \phi$
    for each restriction F_i **do**
      begin
        $F' = F' \cup F_i$
      end
    compute F'^+;
    if $(F'^+ = F^+)$ **then** return (true)
      else return (false);
- The procedure for checking dependency preservation takes exponential time to compute F^+ and $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

Dependency Preservation (3): Example

- **R** (*A, B, C, D, E, F*)
  F = $\{A \rightarrow BCD, A \rightarrow EF, BC \rightarrow AD, BC \rightarrow E, BC \rightarrow F, B \rightarrow F, D \rightarrow E\}$
- Decomposition: **R1**(A, B, C, D) **R2**(B, F) **R3**(D, E)
  - $A \rightarrow BCD$, $BC \rightarrow AD$ are preserved on table R1
  - $B \rightarrow F$ is preserved on table R2
  - $D \rightarrow E$ is preserved on table R3
  - We have to check whether the remaining FDs: **A**→**E**, **A**→ **F**, **BC**→**E**, **BC**→ F are preserved or not.

R1	R2	R3
F_1={**A** → *ABCD*, **B** → *B*, **C** → *C*, **D** → *D*, **AB** → *ABCD*, **BC** → *ABCD*, **CD** → *CD*, **AD** → *ABCD* **ABC** → *ABCD*, **ABD** → *ABCD*, **ACD** → *ABCD* **BCD** → *ABCD*}	F_2={**B** → *BF*, **F** → *F*}	F_3={**D** → *DE*, **E** → *E*}

  - $F' = F_1 \cup F_2 \cup F_3$.
  - Checking for: **A** → *E*, **A** → *F* in F'^+
    - ▷ $A \rightarrow D$ (from R1), $D \rightarrow E$ (from R3) : **A**→ **E** (By Transitivity)
    - ▷ $A \rightarrow B$ (from R1), $B \rightarrow F$ (from R2) : **A**→ **F** (By Transitivity)
  - Checking for: **BC** → *E*, **BC** → *F* in F'^+
    - ▷ $BC \rightarrow D$ (from R1), $D \rightarrow E$ (from R3) : **BC**→ **E** (By Transitivity)
    - ▷ $B \rightarrow F$ (from R2) : **BC**→ **F** (By Augmentation)

Hence all dependencies are preserved.

Dependency Preservation (5): Testing

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \cdots, R_n$ we apply the following test (with attribute closure done with respect to F)
  - $result = \alpha$
    while (changes to result) do
      for each R_i in the decomposition
        $t = (result \cap R_i)^+ \cap Ri$
        $result = result \cup t$
  - 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 $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

Q.1 Let us consider the relation R(A,B,C,D,E) with the following functional dependencies set : $\mathcal{F} = \{AB \rightarrow C, C \rightarrow D, B \rightarrow E\}$.
Let R be decomposed into R_1 and R_2. Check whether the following decomposition is lossless or lossy join decomposition.

1. $R_1(A, B, C)$ and $R_2(D, E)$
2. $R_1(A, B, C, D)$ and $R_2(B, E)$

(1) $R_1 \cup R_2 = ABCDE = R$

$R_1 \cap R_2 = \phi$ ✗

→ Lossy

(2) $R_1 \cup R_2 = ABCE = R$ ✓

$R_1 \cap R_2 = B \neq \phi$ ✓

$R_1 \cap R_2 = B$

Let $B^+ = BE = R_2$

$R_1 \cap R_2 \rightarrow R_2$ ✓

→ Lossless

USE GIVEN DEPENDENCIES TO CHECK FOR SUPERKEY

Q.2 Consider the relation $\mathbf{R}(A,B,C,D,E,G)$ and the functional dependencies set $\mathcal{F} = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$.
Let R be decomposed into $R_1(A, B, C)$, $R_2(A, C, D, E)$ and $R_3(A, D, G)$. Check whether the decomposition is lossless or lossy join decomposition.

① $R_1 \cup R_2 \cup R_3 = ABCDEG = R$ ✓

② $R_1 \cap R_2 = AC \neq \phi$ ✓

③ $R_1 \cap R_2 \rightarrow R_1$ or R_2 ✓

$AC^+ = ACB = R_1$

Let $R_{12} = R_1 \cup R_2 = ABCDE$

$R_{12} \cap R_3 = AD \neq \phi$ ✓

$R_{12} \cap R_3 \rightarrow R_{12}$ or R_3

$AD^+ = ADE$

$ADE^+ = ADEG$

$R_{12} \cap R_3 \rightarrow R_3$ ✓

$\rightarrow$ lossless

Note: In certain books, a **minimal cover** is referred to as that canonical cover which has single attributes on the RHS.
Eg: $A \to BC$ is part of a canonical cover but not a minimal cover.
$A \to B, A \to C$ belong to a minimal cover.

CLOSURE OF ALL IN LHS

$A \to BC,\ CD \to E,\ B \to D,\ E \to A$

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

$C \to E$, $D \to E$

$A \to D$

$A^+ = ABCDE$
$(CD)^+ = CDEAB = ABCDE$
$B^+ = BD$
$E^+ = EABCD = ABCDE$

$A \to BC,\ CD \to E,\ B \to D, E \to A$

$B \to A,\ D \to A,\ AB \to D$

$B \to A \Rightarrow BB \to AB,\ AB \to D$

$\Rightarrow B \to D \Rightarrow B \to D$
$AB \to D$

$B \to A,\ D \to A \quad B \to D$

$B \to D, D \to A \Rightarrow B \to A$

$B \to D, D \to A$

Consider a schema $R(A, B, C, D, E, F, G)$ that satisfies the following FDs:

$$\{A \to B, BC \to DE, AEF \to G\}$$

Is the FD: $ACF \to DG$ implied by this set?

Idea credit: Date, C. J., An introduction to database systems. Edition 8. Pearson Education India.

$(ACF)^+$

Result = ACF
= ACFB (A → B)
= ACFBDE (BC → DE)
= ACFBDEG (AEF → G)

Prove that AG is a superkey.

Idea credit: Silberschatz, A., Korth, H. F., & Sudarshan, S. D

$(AG)^+ =$

ALL ATTRIBUTES IN R

Steps to determine Candidate key

- Determine all the attributes which are not present on RHS of any functional dependency.
- These attributes are always a part of every candidate key. This is because they can not be determined by other attributes.

eg R = (A B C D) , F.D = {A → B, B → C, C → D}

Consider a relation **R**(A, B, C, D, E) and a set of functional dependency on **R**:
F $= \{AB \rightarrow C, DE \rightarrow B, CD \rightarrow E\}$. Find out all the candidate keys and prime attributes?

$\rightarrow (AD)^+ = \{AD\}$

1. $(ADB)^+ = ADB$
   $= ADBC$
   $= ADBCE = R$

2. $(ADC)^+ = ADC$
   $= ADCE$
   $= ADCEB = R$

3. $(ADE)^+ = ADE$
   $= ADEB$
   $= ADEBC = R$

C.K

3

→ ADB

→ ADC

→ ADE

Prime Attributes :- A, B, C, D, E

Example 3

Consider the relation scheme R(E, F, G, H, I, J, K, L, M, N) and the set of functional dependencies: **F** = {{E, F} → G, F → {I, J}, {E, H} → {K, L}, K → M, L → N}. Check which of the following is a super key?

- EFH
- MNEFH
- (GHIK)$^+$ = GHIKM
- (JKLMN)$^+$ ≠ R
- EFGH
- KLMNEFGH

C.K

→ (EFH)$^+$ = EFH
= EFHG
= EFHGIJ
= EFHGIJKL
= EFHGIJKLM
= EFHGIJKLMN = R

FIND THE CANDIDATE KEY OR COMPUTE CLOSURE OF EACH SUPERKEY OPTION

Testing Dependency Preservation

- **R**(A,B,C,D) : **F** = {A → B, B → C, C → D}
- *Decomp* = **R1**(A,B) **R2**(B,C) **R3**(C,D)

ONLY THOSE DEPENDENCIES THAT CONTAIN A AND B ONLY

R1 (A,B) : {A→B} = F1

R2 (B,C) : {B→C} = F2

R3 (C,D) : {C→D} = F3

Cond. for preservation of dependency is:

F1 ∪ F2 ∪ F3 = F

Decomposition is dependency preserving.

Testing Dependency Preservation: Using Closure Set of FD (Exp. Algo.)

- **R** (A, B, C, D)
  F = {A → B, B → C, C → D, D → A}
- Decomposition: **R1**(A, B) **R2**(B, C) **R3**(C, D)
  - A → B is preserved on table R1
  - B → C is preserved on table R2
  - C → D is preserved on table R3
  - We have to check whether the one remaining FD: D→A is preserved or not.

X(ABC)
F+ : A+ = {A, B} AB+ ABC+ = ABC
B+ : {B, C} Bc+
C+ : {C, A} Ac+

R1	R2	R3
F_1^+ = { }	F_2^+ = { }	F_3^+ = { }

- $F' = F_1^+ \cup F_2^+ \cup F_3^+$.
- Checking for: **D** → *A* in F'^+

Testing Dependency Preservation: Using Closure Set of FD (Exp. Algo.)

- **R** (A, B, C, D)
  F $= \{A \to B, B \to C, C \to D, D \to A\}$
- Decomposition: **R1**(A, B) **R2**(B, C) **R3**(C, D)
  - $A \to B$ is preserved on table R1
  - $B \to C$ is preserved on table R2
  - $C \to D$ is preserved on table R3
  - We have to check whether the one remaining FD: **D**→**A** is preserved or not.

R1	R2	R3
$F_1^+ = \{ A \to B, B \to A \}$	$F_2^+ = \{ B \to C, C \to B \}$	$F_3^+ = \{ C \to D \quad D \to C \}$

- $F' = F_1^+ \cup F_2^+ \cup F_3^+$.
- Checking for: $\mathbf{D} \to A$ in F'^+

D^+ on $F'^+ = \{A, B, C, D\}$, $\therefore$ $D \to A$ in F'^+.

Testing Dependency Preservation

- $\mathbf{R}(A,B,C,D,E)$: $\mathbf{F} = \{A \rightarrow BC, C \rightarrow DE, D \rightarrow E\}$ $C \rightarrow E$
- *Decomp* = $\mathbf{R1}(A,B,C,D)$ $\mathbf{R2}(D,E)$

F_1^+: $A^+ = \{A,B,C,D\}$

$B^+ = \{B\}$

$C^+ = \{C,D\}$

$D^+ = \{D\}$

$AB^+ = \{A\,B\,C\,D\}$

$BC^+ = \{B\,C\,D\}$

$CD^+ = \{C\,D\}$

$DA^+ = \{A\,B\,C\,D\}$

$AC^+ = \{A\,B\,C\,D\}$

$BD^+ = \{B\,D\}$.

$BCD^+ = \{B\,C\,D\}$.

F_2^+

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

$E^+ = \{E\}$

$F_1^+ \cup F_2^+ = F$

Check whether C^+ has E.

1) $C \rightarrow D$ 2) $D \rightarrow E$ $\therefore C \rightarrow E$