University of Chittagong

Department of Computer Science & Engineering

Database Systems Lab

Name of the assignment:

Assignment 7: Chapter 7 Exercise

CSE 414

Database Systems

Submitted By:
Debashish Chakraborty
ID: 23701034

Submitted To:
Dr. Rudra Pratap Deb Nath
Associate Professor

July 7, 2025

Problem 7.8: Algorithm Efficiency Analysis

Question

7.8: Consider the algorithm in Figure 7.19 to compute α^+. Show that this algorithm is more efficient than the one presented in Figure 7.8 (Section 7.4.2) and that it computes α^+ correctly.

```
result := ∅;
/* fdcount is an array whose ith element contains the number
   of attributes on the left side of the ith FD that are
   not yet known to be in α+ */
for i := 1 to |F| do
   begin
     let β → γ denote the ith FD;
     fdcount [i] := |β|;
   end
/* appears is an array with one entry for each attribute. The
   entry for attribute A is a list of integers. Each integer
   i on the list indicates that A appears on the left side
   of the ith FD */
for each attribute A do
   begin
     appears [A] := NIL;
     for i := 1 to |F| do
       begin
         let β → γ denote the ith FD;
         if A ∈ β then add i to appears [A];
       end
   end
addin (α);
return (result);

procedure addin (α);
for each attribute A in α do
   begin
     if A ∉ result then
       begin
         result := result ∪ {A};
         for each element i of appears[A] do
           begin
             fdcount [i] := fdcount [i] − 1;
             if fdcount [i] := 0 then
               begin
                 let β → γ denote the ith FD;
                 addin (γ);
               end
           end
       end
   end
```

Figure 7.18 An algorithm to compute α^+.

Solution:
The given algorithm's goal is to compute α^+.

1. Initially, all attributes in α are added to the result.
2. For each FD $\beta \to \gamma$, the algorithm tracks how many attributes from β are still missing using an array `fdcount`.
3. `appears[A]` lists all FDs where attribute A appears in the LHS.
4. When an attribute A is added to the result, the algorithm checks all FDs waiting for A via `appears[A]`.
5. If all attributes in the LHS β of some FD are now in the result (`fdcount[i]` becomes 0), the corresponding RHS γ is recursively added to the result.

Why is it correct?
Every attribute A added to the result has a valid derivation $\alpha \to A$ based on FDs. This is ensured because A is only added when there is a dependency $\beta \to \gamma$ such that $\beta \subseteq \alpha^+$ and $A \in \gamma$.

If $A \in \alpha^+$, then A will be added to the result. This is provable by induction on the length of the proof of $\alpha \to A$ using Armstrong's axioms:

- **Base Case:** If $A \in \alpha$, it is added in the initial call to `addin`.
- **Inductive Step:** Suppose $\alpha \to A$ is proven in $n + 1$ steps. Then there must be some dependency $\beta \to \gamma$ with $\beta \subseteq \alpha^+$ and $A \in \gamma$. By inductive hypothesis, all of β has already been added to the result, triggering the addition of γ, and thus A.

Efficiency Analysis

Let us now compare the performance of the two algorithms.

In **Figure 7.19**, each FD is scanned exactly once in the initialization phase to compute `fdcount` and `appears`. The recursive calls to `addin` process attributes only once and process only relevant FDs using `appears[A]`. So every FD and attribute is processed only when needed.

In contrast, the algorithm in **Figure 7.8** repeatedly scans the entire FD set in a loop until no new attributes are added to the closure. In the worst case, this could result in $O(|F|^2)$ time complexity if each FD triggers a new iteration.

Complexity Comparison

Algorithm in Figure 7.19: $O(|F| + |R|)$ (linear in the size of FDs and attributes)
Algorithm in Figure 7.8: $O(|F|^2)$ (may scan all FDs multiple times)

Problem 7.26: Functional Dependency Rule Soundness

Question

7.26: Consider the following proposed rule for functional dependencies: If $\alpha \to \beta$ and $\gamma \to \beta$, then $\alpha \to \gamma$. Prove that this rule is not sound by showing a relation r that satisfies $\alpha \to \beta$ and $\gamma \to \beta$, but does not satisfy $\alpha \to \gamma$.

Solution:

We can prove that this rule is not sound by providing an example. Consider the following relation R:

α	γ	β
A	X	100
B	Y	200
B	Z	200

In this relation:

- $\alpha \to \beta$ holds: Each value of α maps to a unique value of β
  - $\alpha = A$ maps to $\beta = 100$
  - $\alpha = B$ maps to $\beta = 200$
- $\gamma \to \beta$ holds: Each value of γ maps to a unique value of β
  - $\gamma = X$ maps to $\beta = 100$
  - $\gamma = Y$ maps to $\beta = 200$
  - $\gamma = Z$ maps to $\beta = 200$
- However, $\alpha \to \gamma$ does **not** hold: The value $\alpha = B$ maps to two different values of γ (both Y and Z)

Therefore, If $\alpha \to \beta$ and $\gamma \to \beta$, then $\alpha \to \gamma$, is not sound.

Problem 7.35: BCNF Decomposition and Lossless Property

Question

7.35: Although the BCNF algorithm ensures that the resulting decomposition is lossless, it is possible to have a schema and a decomposition that was not generated by the algorithm, that is in BCNF, and is not lossless. Give an example of such a schema and its decomposition.

Solution:

Consider the schema $R = (A, B, C, D)$ with the following set F of functional dependencies:

$F := \{AB \to C, CD \to A\}$

Consider the following decomposition of R:

$$R_1 = (A, B, C) \tag{1}$$
$$R_2 = (A, C, D) \tag{2}$$

BCNF Verification

For $R_1 = (A, B, C)$:

Functional dependencies projected onto R_1: $F_1 = \{AB \to C\}$

We need to check if AB is a superkey for R_1: $(AB)^+ = \{A, B, C\}$ (within R_1)

Since AB determines all attributes in R_1, it is a superkey. Therefore, R_1 is in BCNF.

For $R_2 = (A, C, D)$:

Functional dependencies projected onto R_2: $F_2 = \{CD \to A\}$

We need to check if CD is a superkey for R_2: $(CD)^+ = \{A, C, D\}$ (within R_2)

Since CD determines all attributes in R_2, it is a superkey. Therefore, R_2 is in BCNF.

Lossless Property Verification

To check if the decomposition is lossless, we use the condition: $(R_1 \cap R_2) \to R_1$ or $(R_1 \cap R_2) \to R_2$

We have: $R_1 \cap R_2 = \{A, B, C\} \cap \{A, C, D\} = \{A, C\}$

For lossless join, we need either:

1. $AC \to ABC$ (which means $AC \to B$), or
2. $AC \to ACD$ (which means $AC \to D$)

From the given FDs $\{AB \to C, CD \to A\}$, we cannot derive $AC \to B$. AC does not determine B.

From the given FDs $\{AB \to C, CD \to A\}$, we cannot derive $AC \to D$. AC does not determine D.

Since neither condition holds, the decomposition is **lossy**.

This example demonstrates that it is possible to have a BCNF decomposition that is not lossless.