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 6, 2025

Contents

Problem 7.8: Algorithm Efficiency Analysis

Question

7.8: Consider the algorithm in Figure 7.18 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.

Solution:

Correctness Proof

The algorithm is correct because:

- If A is added to result then there is a proof that $\alpha \rightarrow A$. To see this, observe that $\alpha \rightarrow \alpha$ trivially, so α is correctly part of result. If $A \notin \alpha$ is added to result, there must be some FD $\beta \rightarrow \gamma$ such that $A \in \gamma$ and β is already a subset of result. (Otherwise fdcount would be nonzero and the if condition would be false.) A full proof can be given by induction on the depth of recursion for an execution of addin, but such a proof can be expected only from students with a good mathematical background.

- If $A \in \alpha^+$, then A is eventually added to result. We prove this by induction on the length of the proof of $\alpha \rightarrow A$ using Armstrong's axioms. First observe that if procedure addin is called with some argument β, all the attributes in β will be added to result. Also if a particular FD's fdcount becomes 0, all the attributes in its tail will definitely be added to result. The base case of the proof, $A \in \alpha \Rightarrow A \in \alpha^+$, is obviously true because the first call to addin has the argument α. The inductive hypothesis is that if $\alpha \rightarrow A$ can be proved in n steps or less, then $A \in$ result. If there is a proof in $n+1$ steps that $\alpha \rightarrow A$, then the last step was an application of either reflexivity, augmentation, or transitivity on a fact $\alpha \rightarrow \beta$ provided in n or fewer steps. If reflexivity or augmentation was used in the $(n+1)^{st}$ step, A must have been in result by the end of the n^{th} step itself. Otherwise, by the inductive hypothesis, $\beta \subseteq$ result. Therefore, the dependency used in proving $\beta \rightarrow \gamma$, $A \in \gamma$, will have fdcount set to 0 by the end of the n^{th} step. Hence A will be added to result.

Efficiency Analysis

To see that this algorithm is more efficient than the one presented in the chapter, note that we scan each FD once in the main program. The resulting array appears has size proportional to the size of the given FDs. The recursive calls to addin result in processing linear in the size of appears. Hence the algorithm has time complexity which is linear in the size of the given FDs.

On the other hand, the algorithm given in the text has quadratic time complexity, as it may perform the loop as many times as the number of FDs, in each loop scanning all of them once.

Complexity Comparison:

$$\text{Figure 7.18 Algorithm:} \quad O(|F|) \tag{1}$$
$$\text{Figure 7.8 Algorithm:} \quad O(|F|^2) \tag{2}$$

where $|F|$ represents the size of the given functional dependencies.

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 a counterexample. Consider the following relation r:

α	γ	β
1	6	7
2	3	5
2	4	5

In this relation:

- $\alpha \to \beta$ holds: Each value of α maps to a unique value of β
  - $\alpha = 1$ maps to $\beta = 7$
  - $\alpha = 2$ maps to $\beta = 5$
- $\gamma \to \beta$ holds: Each value of γ maps to a unique value of β
  - $\gamma = 6$ maps to $\beta = 7$
  - $\gamma = 3$ maps to $\beta = 5$
  - $\gamma = 4$ maps to $\beta = 5$
- However, $\alpha \to \gamma$ does **not** hold: The value $\alpha = 2$ maps to two different values of γ (both 3 and 4)

This counterexample proves that the proposed rule is not sound.

Mathematical Verification:

Let $R = \{\alpha, \gamma, \beta\}$ and $F = \{\alpha \to \beta, \gamma \to \beta\}$.
The proposed rule states: $F \models \alpha \to \gamma$
However, our counterexample relation r shows that:

$$r \models F \text{ but } r \not\models \alpha \to \gamma$$

Therefore, $F \not\models \alpha \to \gamma$, proving the rule is unsound.

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:

Take the schema $R = (A, B, C, D, E)$ given on Practice Exercise 1. Assume the following set F of functional dependencies holds:

$$F := \{A \to BC, CD \to E, B \to D, E \to A\}$$

Take the decomposition of R given on Exercise 7.29:

$$R_1 = (A, B, C) \tag{3}$$
$$R_2 = (C, D, E) \tag{4}$$

BCNF Verification

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

Functional dependencies projected onto R_1:

$$F_1 = \{A \to BC, A \to B, A \to C\}$$

Since A is a superkey for R_1 (as $A^+ = \{A, B, C\}$ within R_1), all functional dependencies have a superkey on the left side. Therefore, R_1 is in BCNF.

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

Functional dependencies projected onto R_2:

$$F_2 = \{CD \to E\}$$

We need to check if CD is a superkey for R_2:

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

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 general condition:

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

We have:

$$R_1 \cap R_2 = \{A, B, C\} \cap \{C, D, E\} = \{C\}$$

Now we check:

- Does $C \rightarrow R_1 = \{A, B, C\}$?

  $C^+ = \{C\}$ (from the given functional dependencies)

  Since $C^+ \not\supseteq \{A, B, C\}$, this condition fails.

- Does $C \rightarrow R_2 = \{C, D, E\}$?

  Since $C^+ = \{C\}$ and $C^+ \not\supseteq \{C, D, E\}$, this condition also fails.

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

This example demonstrates that it is possible to have a BCNF decomposition that is not lossless, even though the BCNF algorithm itself guarantees lossless decomposition. The key insight is that not all BCNF decompositions are generated by the algorithm, and manually created BCNF decompositions may not preserve the lossless property.