****

**Group Project 4**

**Alex Salazar-Almaraz, Jordan Nguyen, Frederick Smeltzer, Danny Ta**

****

CS/DSA-4513 – Database management systems

instructor:

Dr. Le Gruenwald

**Relational Schema:**

R (A, B, C, D, E, F, G)

**Functional Dependencies:**

SetOfFDs = {A -> B, B -> AC, F -> ACDE, ADE -> FG}

**Non-prime attributes:**

C, G

**Problem 1:**

1. Candidate keys (Highlighted):

A+ = ABC

B+ = ABC

C+ = C

D+ = D

E+ = E

F+ = ABCDEFG

G+ = G

AB+ = ABC

AC+ = ACB

AD+ = ADBC

AE+ = AEBC

BD+ = BDAC

BE+ = BEAC

DE+ = DE

ABD+ = ABDC

ABE+ = ABEC

BDE+ = ABCDEFG

ADE+ = ABCDEFG

ABDE+ = ABCDEFG

Our non-prime attributes are those that are not present in our candidate keys. The only two not present are C and G.

1. Normal Forms:
   1. 1NF – Each of the attributes is atomic, meaning that each is not divisible, so it will be in 1NF and is satisfied by the functional dependencies.
   2. 2NF – To be in second normal form, it must already be in 1NF but also must have no non-prime attributes with partial dependencies on the primary key.

Candidate keys – F, BDE, ADE

Non-prime attributes – C, G

G:

G is fully dependent on each of the candidate keys. We don’t see G in any subset of the candidate keys

F+ = ABCDEFG

BDE+ = ABCDEFG

B+ = ABC

D+ = D

E+ = E

BD+ = BDAC

BE+ = BEAC

ADE+ = ABCDEFG

A+ = ABC

D+ = D

E+ = E

AD+ = ADBC

AE+ = AEBC

C:

C is partially dependent, thus breaking the 2NF. We can see that subsets of each of the candidate keys consist of the attribute C

BDE+ = ABCDEFG

B+ = ABC

D+ = D

E+ = E

BD+ = BDAC

BE+ = BEAC

ADE+ = ABCDEFG

A+ = ABC

D+ = D

E+ = E

AD+ = ADBC

AE+ = AEBC

* 1. 3NF – To be in third normal form, it must already be in 2NF but also must have no attributes with transitive dependencies on the primary key. This is not satisfied because we do not have 2NF.
  2. BNCF – To be in Boyce-Codd Normal Form it must be in 3NF but has stricter terms. For any non-trivial functional dependency, X -> A, X must be a super-key. This is not satisfied because we do not have 2NF or 3NF.

1. Decomposition Algorithm:

R (A, B, C, D, E, F, G)

F = {A -> B,

B -> AC,

F -> ACDE,

ADE -> FG}

ADE and F are candidate keys. We need to decompose A and B where

Result := (result- Ri) U (Ri-B) U (a, B)

Result = {R}

R1 = (ABDEFG)

R2 = (A,C)

R = { } U { R1 = (ABDEFG)} U { R2 = (A,C)}

We have R1 and R2

We now need to check to see if any subset of R1 or R2 violates the 2NF. This algorithm guarantees that R1 is in 2NF, but what about R2? (Not sure of this part, how to take it further)

1. Lossless:

Using the decomposition algorithm, we know that we are guaranteed a lossless join. If one of the following dependencies is in F+, R1 ∩ R2 → R1 or R1 ∩ R2 → R2 then we know that there was a lossless join. We can see that this is true by combining the attributes from both R’s: (ABDEFG) + (AC) = R

1. Dependency:

Using the decomposition algorithm, we are not guaranteed dependency, so we must check to see if it has been preserved.

R1 = (ABDEFG)

R2 = (AC)

R1 ∩ R2 = {A} and A -> AB in F+, i.e., R1 ∩ R2 -> R1 in F+

This is not dependency preserving as we cannot check:

B -> AC

F -> ACDE

ADE -> FG

Without computing R1 |x| R2

**Problem 2:**

**Problem 3:**