20BCE154

20bce103

20bce137

DBMS INNOVATIVE ASSIGNMENT

CANONIAL COVER

When updating a database, the system's role is to ensure that no current functional dependencies are broken in the process. In the event that the new database state violates functional dependencies, the system must be rolled back.

Irreducible functional relationships or a canonical cover FD is a reduced version of FD with a comparable closure to the original FD set.

#### **Important terms:**

1. Extraneous attributes: If we can delete an attribute of a functional dependency without modifying the closure of the set of functional dependencies, it is said to be unnecessary.
2. Canonical cover: Canonical cover Fc is a set of functional dependencies F such that the following properties are not violated:

1) All dependencies in Fc are logically implied by F.

2) All dependencies in F are logically implied by Fc.

3) There are no extraneous attributes in Fc's functional dependencies.

4) Each functional dependency's left side in Fc is distinct. There are no two dependencies ɑ1→β1 and ɑ2→β2 in such that ɑ1→ɑ2.

#### **Finding Canonical Cover**

Following is an Algorithm to determine the Canonical cover of set F:

repeat

1. Replace any dependencies in ɑ1→β1 and ɑ1→β2. with ɑ1→β1β2 using the union rule.
2. With an extraneous attribute, either in ɑ or β find a functional dependency ɑ→β.
3. Delete any extraneous attribute found in ɑ→β.

until F does not change.

**Example 1:**

Consider the following set of functional dependencies:

F = {

X→YZ

Y→Z

X→Y

XY→Z

}

According to the steps mentioned above:

1. The two functional dependencies

X→YZ

X→Y

with same attributes on the left can be combined to get

X→YZ.

Now, the new set is:

F = {

X→YZ

Y→Z

XY→Z

}

2. XYZ is an extraneous attribute because we get the same closures even after removing it from the set. It is since Y→Z is already part of F.

The new set F comes out to be:

F = {

X→YZ

Y→Z

}

3. X→Y is logically suggested by X→Y and Y→Z, yet Z is an extraneous X→YZ attribute (transitivity).

F = {

X→Y

Y→Z

}

4. F does not change anymore after this step. The required Canonical cover is,

F = {

X→Y

Y→Z

}

Example 2:

Consider another set of functional dependencies:

F={

A→BC

CD→E

B→D

E→A

}

According to the algorithm:

1. Each functional dependency in F has a distinct left side.
2. There are no extraneous attributes on the left or right sides of any functional dependency (Checked by applying the definition of extraneous attributes on every functional dependency).
3. Hence, the canonical cover Fc is equal to F.

How to verify if a set of f.d’s F canonically covers a set of f.d’s G?

Consider the two sets of functional dependencies:

F={

A→B

AB→C

D→AC

D→E

}

G={

A→BC

D→AB

}

We must now determine whether one of these f.d.’s canonically covers the other set of f.d.’s. We must decide if F canonically covers G, G canonically covers F, or neither of the two canonically covers the other.

1. Make a singleton on the right side. This indicates that all of the characteristics on the right side of the f.d. arrow must be singletons.

Functional interdependence The functional dependencies D→A and D→C are separated from D→AC.

F={

A→B

AB→C

D→A

D→C

D→E

}

2. Remove any extraneous attributes that aren't necessary.

Consider any XY→Z functional dependence. If X can determine Z by itself, then the attribute Y is extraneous and can be deleted. As can be seen, unnecessary attributes can only exist in functional relationships with many attributes in the LHS.

Take the functional dependency AB→C as an example.

To determine whether any of them are superfluous, we must first find the closures of A and B.

[A]+ = AB

[B]+ = B

As can be seen, B can be deduced from A. This means that B can be removed from the functional dependency AB→C.

F={

A→B

A→C

D→A

D→C

D→E

}

Remove all functional dependencies that are no longer needed.

Check all f.d.’s one by one to verify if eliminating an f.d. X→Y still allows us to deduce Y from X using another f.d. We are testing and checking whether Y is a component of the closure without using the f.d. A more formal way to describe this is to find [X]+ without using the f.d. If this is the case, the f.d. is no longer necessary.

When looking for the f.d. D → C, we notice that the closure of D contains C even after it has been hidden. By combining two other f.d.’s D→A and A→C, we can get C from D. As a result, →C is redundant.

Now, repeat the steps for G.

1. Make a singleton on the right side. This indicates that all of the characteristics on the right side of the f.d. arrow must be singletons.

G = {

A→B

A→C

D→A

D→B

}

2. Remove any extraneous attributes that aren't necessary. There can be no unnecessary attributes because the RHS of all f.d's has only one attribute.

3. Remove all functional dependencies that are no longer needed.

We can see that the f.d. D→B is redundant because it can be obtained by combining two other f.d .'s, D→A and A→B. By looping over all f.d's and checking the closure of the LHS in all cases, we can see that the f.d. D→B is redundant because it can be obtained by combining two other f.d .'s, D→A and A→B.

G={

A→B

A→C

D→A

}

As we can see that all G’s f.d.’s are already covered in F, we conclude that F covers G.