CSC3170 Assignment2 Wang Jiaju 121090544

1.

(a) $A \rightarrow \emptyset$ is trivial because any set of attributes functionally determines the empty set. Since there are no attributes to determine on the right side, it's trivially satisfied.

Example:

Suppose A is BookID with instances: {1, 2, 3}, and we can refer no information from this BookID. “No information” here is a subset of “BookID”, and by the definition, we have $A \rightarrow \varphi$ is trivial.

(b) $\emptyset \rightarrow A$ is non-trivial because having no attribute on the left side cannot determine any attribute. This doesn't hold by the definition of functional dependency unless A is also empty, but it's given that $A \neq \emptyset$.

Example:

Suppose A is still BookID with instances: {1, 2, 3}, and φ means no information. Since A is not a subset of φ, by the definition of “trivial FD”, we have $A \rightarrow \varphi$ is a nontrivial FD.

(c) $\emptyset \rightarrow \emptyset$ is trivial because the empty set trivially determines itself.

Example:

φ refers to “no information”. If we have no information, then we can refer to nothing. Also, by the definition of “Trivial FD”, we have φ is a subset of φ, and as a result, this is a trivial FD.

2. To show $|F^*| \leq 2^{(2n)}$, we must acknowledge that F* is the closure of F, including all direct and indirect dependencies that can be derived from F. Since each attribute can either be present or absent in a functional dependency, there are 2^n possible combinations of attributes on the left side and 2^n combinations on the right. Thus, the number of possible functional dependencies cannot exceed $2^n * 2^n = 2^{(2n)}$.

3. A canonical cover is a minimal set of functional dependencies that is equivalent to F. For the given set of functional dependencies F:

$A \rightarrow BC$; $B \rightarrow AC$; $C \rightarrow AB$

Four different canonical covers could be the following, assuming no further simplification from attribute closure:

$\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

{A → C, B → A, C → B}

{A → C, B → C, C → AB}

{A → B, B → AC, C → B}

4.

A functional dependency, denoted as X→ Y, within the context of a relation R, asserts that for any two tuples t1 and t2 inR, if t1[X] = t2[X]\), then t1[Y] = t2[Y]. This means that the value of X determines the value of Y.

A multivalued dependency, denoted as X →→ Y, within a relation R, asserts that for any two tuples t1 and t2 in R where t1[X] = t2[X], there exist tuples t3 and t4 in R such that:

t1[X] = t3[X] = t2[X] = t4[X]

t1[Y] = t3[Y] and t2[Y] = t4[Y]

t3[Z] = t2[Z] and t4[Z] = t1[Z], for all attributes Z in R that are not in X or Y.

The next, given a relation R and a functional dependency X →Y, we need to show that if X →Y holds, then X →→Y also holds.

1. Given X →Y, for any two tuples t1 and t2 in R where t1[X] = t2[X], it's given by FD that t1[Y] = t2[Y].

2. Since X →Y and assuming t1[X] = t2[X], we can construct tuples t3 and t4 from t1 and t2 as follows:

t3 is a tuple identical to t1, and since t1[Y] = t2[Y] due to the FD, t3[Y] can be set to t2[Y] and vice versa.

t4 would be constructed similarly to t3, ensuring that for all attributes Z in R that are not in X or Y, t3[Z] = t2[Z] and t4[Z] = t1[Z] hold by the definition of FD.

3. With t3 and t4 constructed as above, all conditions for X →→Y are satisfied:

t1[X] = t3[X] = t2[X] = t4[X] by construction based on the equivalence provided by the FD.

(t1[Y] = t3[Y] and t2[Y] = t4[Y] are ensured by the FD and the way t3 and t4 are constructed.

For all other attributes Z, t3[Z] = t2[Z] and t4[Z] = t1[Z], maintaining the nonX and nonY attribute values across the tuples as required.

Therefore, given a functional dependency X →Y in a relation R, it implies a multivalued dependency X →→ Y, since we can construct the necessary tuples to satisfy the conditions for MVD based on the determinacy guaranteed by FD. This shows how FDs naturally satisfy the formal definition of MVDs within relational database theory.

5:

(i) Schema of RelationX: Since RelationX is a join of Order and Order-Item on the common attribute 'O#', the schema will include all attributes from both relations. The resulting schema will be:

RelationX (O#, Odate, Cust#, Total_amount, I#, Qty_ordered, Total_price, Discount%)

(ii) Primary Key: (O#, I#)

(iii) Functional dependencies of RelationX:

The assumptions are based on typical business rules for orders:

'O#' uniquely determines 'Odate', 'Cust#', and 'Total_amount'.

'I#' does not determine any other attributes uniquely because it's not unique across orders.

The combination of 'O#' and 'I#' determines 'Qty_ordered', 'Total_price', and 'Discount%'.

Based on these assumptions, the functional dependencies would be:

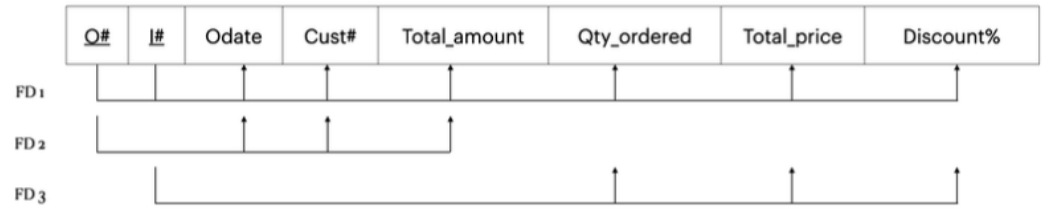
O# → Odate, Cust#, Total_amount

(O#, I#) → Qty_ordered, Total_price, Discount%

(iv) Normalization of RelationX:

To determine if it's in 2NF, we check for partial dependencies. Since 'Odate', 'Cust#', and 'Total_amount' are only dependent on 'O#' and not on 'I#', and these are non-prime attributes (not part of a candidate key), RelationX is not in 2NF.

To determine if it's in 3NF, we first need to be in 2NF. Since it's not in 2NF, it's also not in 3NF.



6:

(i) `{Model#}`: This cannot be a candidate key alone if different models can have different prices in different years or be produced in different plants in different colors.

`{Model#, Year}`: This could be a candidate key if we assume that a model's price only changes per year and not by production plant or color.

`{Model#, Color}`: This cannot be a candidate key because the same model could be made in different plants in the same color but at different prices.

(ii) The relation is in 3NF if every non-prime attribute is fully functionally dependent on every key of the relation and there are no transitive dependencies.

To be in BCNF, for every functional dependency (X → Y), X should be a superkey.

Based on the functional dependencies provided:

Model# → Manuf_Plant; Model#, Year → Price; Manuf_Plant → Color

If `{Model#, Year}` is the candidate key, then `Model# → Manuf_Plant` is a violation of BCNF because `Model#` is not a superkey (it's part of a candidate key). Hence, the relation is not in BCNF.

(iii) Yes. A decomposition is lossless if, when the decomposed tables are joined, they result in the original table without any spurious tuples. Using the dependency preservation and original functional dependencies, we can join R1 and R2 on `Model#` to see if we can reconstruct the original relation without loss.

The join condition is on `Model#`, which is common to both R1 and R2, and since `Model#` is part of a candidate key, this decomposition should be lossless as it preserves the functional dependencies and the ability to reconstruct the original relation.

Therefore, the decomposition into R1 and R2 is lossless.