**ALGORITHMS FOR RELATIONAL DATABASE SCHEMA DESIGN**

Samyakth talked about what a Relational decomposition from a Universal Relation is.

Now we are going to talk about the two algorithms for creating a relational decomposition from a universal relation.

The first algorithm decomposes a universal relation into dependency preserving 3-NF relations that also possess the non-additive join property.

The second algorithm decomposes a universal relation into BCNF schema that possess the non-additive join property.

It is not possible to design an algorithm to produce BCNF relations that satisfy both dependency preservation and non-additive join decomposition.

This is because these two properties are in conflict with each other.

To help you understand about the conflict between two properties, has anyone seen a movie called i,ROBOT.

There are three laws of robotics.

1) A robot cannot harm a human-being.

2) A robot must obey any order given by a human-being.

You can see the problem here, what if a human-being ordered the robot to kill another human-being.

This movie revolves around a robot that defies this law. You know what they say, the laws are made to be broken.

3) A robot can defend itself.

These two properties come into conflict with each other because,

The dependency preservation property states that all the dependencies in the relation must be preserved.

The non-additive join decomposition property requires that the resulting relation from a join operation between two tables may not be the sum of the individual rows in each table.

This means that some rows may appear multiple times in the resulting relation if there are multiple occurrences of a given value in one of the tables being joined.

**Dependency-Preserving and Non-additive (lossless) join decomposition into 3NF schemas**

**Input:** A universal relation R and a set of functional dependencies F on the attributes of R.

1. Find a minimal cover G for F (use Algorithm 15.2).
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes {X ∪ {A1} ∪ {A2} … ∪ {Ak} }, where X → A1, X → A2, … , X → Ak are the only dependencies in G with X as lefthand side (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R. (Algorithm 15.2(a) may be used to find a key.)
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S.

**Ex**: Consider the following universal relation:

**U** (Emp\_ssn, Pno, Esal, Ephone, Dno, Pname, Plocation)

The following dependencies are present:

**FD1**: Emp\_ssn → {Esal, Ephone, Dno}

**FD2**: Pno → { Pname, Plocation}

**FD3**: Emp\_ssn, Pno → {Esal, Ephone, Dno, Pname, Plocation}

Minimal cover G: {Emp\_ssn → Esal, Ephone, Dno; Pno → Pname, Plocation}

The second step of Algorithm 15.4 produces relations R1 and R2 as:

**R1** (Emp\_ssn, Esal, Ephone, Dno)

**R2** (Pno, Pname, Plocation)

In step 3, we generate a relation corresponding to the key {Emp\_ssn, Pno} of U. Hence, the resulting design contains:

**R1** (Emp\_ssn, Esal, Ephone, Dno)

**R2** (Pno, Pname, Plocation) R3 (Emp\_ssn, Pno)

This decomposition algorithm achieves dependency preservation, non-additive (lossless or non-lossy design) join property and is in 3-NF.

**Non-additive join decomposition into BCNF schemas**

**Input**: A universal relation R and a set of functional dependencies F on the attributes of R.

1. Set D := {R} ;
2. While there is a relation schema Q in D that is not in BCNF do

{

choose a relation schema Q in D that is not in BCNF;

find a functional dependency X → Y in Q that violates BCNF;

replace Q in D by two relation schemas (Q − Y) and (X ∪ Y); } ;

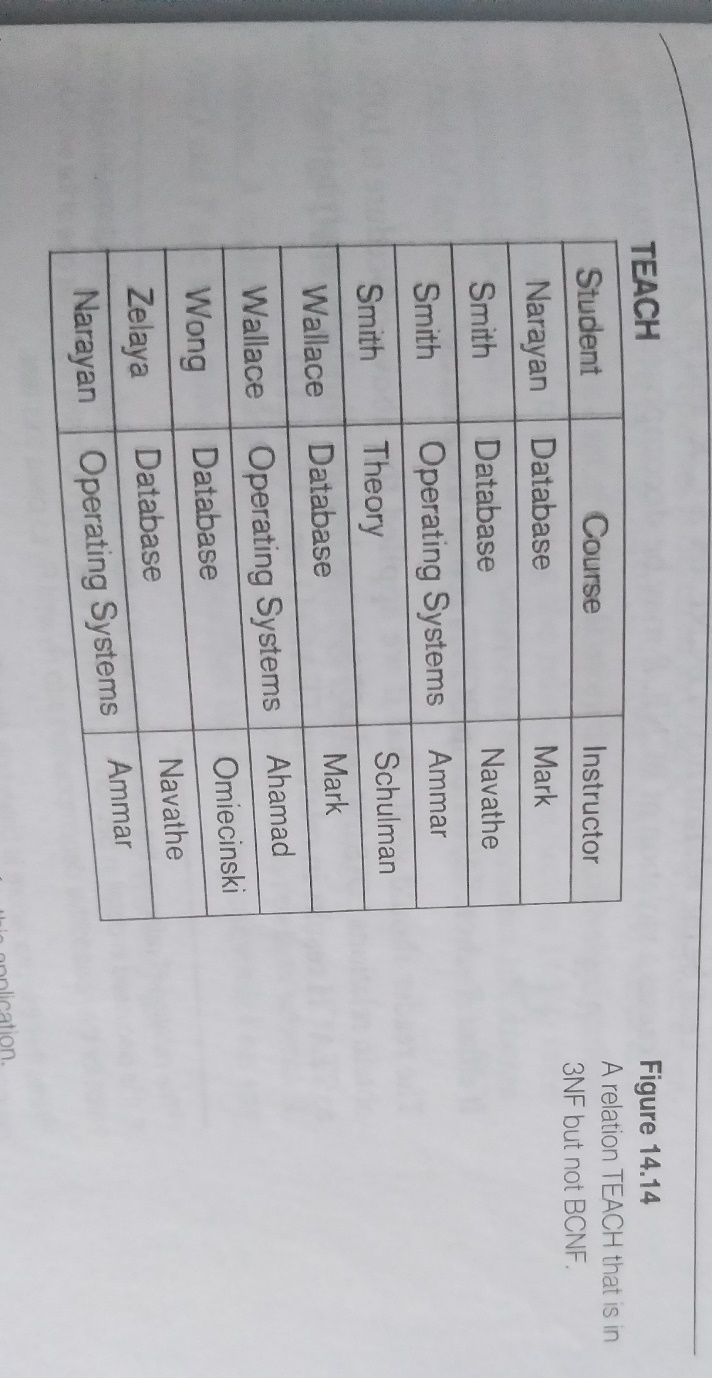
Each time through the loop, we decompose one relation schema Q that is not in BCNF into two relation schemas. At the end of the loop all relation schemas in D will be in BCNF.

Ex: We illustrated the application of this algorithm to the TEACH relation schema.

It is decomposed into TEACH1() and TEACH2() because the dependency Instructor->Course violates BCNF.

FD1: {Student, Course} -> Instructor

FD2: Instructor -> Course



**ABOUT NULLS, DANGLING TUPLES, AND ALTERNATIVE RELATIONAL DESIGNS**

We must carefully consider the problems associated with NULLs when designing a relational database schema. There is no fully satisfactory relational design theory as yet that includes NULL values. One problem occurs when some tuples have NULL values for attributes that will be used to join individual relations in the decomposition. To illustrate this, consider the database shown in Figure 15.2(a), where two relations EMPLOYEE and DEPARTMENT are shown. The last two employee tuples— ‘Berger’ and ‘Benitez’—represent newly hired employees who have not yet been assigned to a department (assume that this does not violate any integrity constraints). Now suppose that we want to retrieve a list of (Ename, Dname) values for all the employees. If we apply the **NATURAL JOIN** operation on EMPLOYEE and DEPARTMENT (Figure 15.2(b)), the two aforementioned tuples will not appear in the result. The **OUTER JOIN** operation, discussed in Chapter 8, can deal with this problem. Recall that if we take the LEFT OUTER JOIN of EMPLOYEE with DEPARTMENT, tuples in EMPLOYEE that have NULL for the join attribute will still appear in the result, joined with an imaginary tuple in DEPARTMENT that has NULLs for all its attribute values. Figure 15.2(c) shows the result.

In general, whenever a relational database schema is designed in which two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential NULL values in foreign keys. This can cause unexpected loss of information in queries that involve joins on that foreign key. Moreover, if NULLs occur in other attributes, such as Salary, their effect on built-in functions such as SUM and AVERAGE must be carefully evaluated.

A related problem is that of dangling tuples, which may occur if we carry a decomposition too far. Suppose that we decompose the EMPLOYEE relation in figure 15.2(a) further into EMPLOYEE\_1 and EMPLOYEE\_2, shown in Figures 15.3(a) and 15.3(b). If we apply the NATURAL JOIN operation to EMPLOYEE\_1 and EMPLOYEE\_2, we get the original EMPLOYEE relation. However, we may use the alternative representation, shown in Figure 15.3(c), where we do not include a tuple in EMPLOYEE\_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE\_2). If we use EMPLOYEE\_3 instead of EMPLOYEE\_2 and apply a NATURAL JOIN on EMPLOYEE\_1 and EMPLOYEE\_3, the tuples for Berger and Benitez will not appear in the result; these are called dangling tuples in EMPLOYEE\_1 because they are represented in only one of the two relations that represent employees, and hence they are lost if we apply an (INNER) JOIN operation.

**Discussion of Normalization Algorithms and Alternative Relational Designs**

One of the problems with the normalization algorithms we described is that the database designer must first specify all the relevant functional dependencies among the database attributes. This is not a simple task for a large database with hundreds of attributes. Failure to specify one or two important dependencies may result in an undesirable design.

It is not always possible to find a decomposition into relation schemas that preserves dependencies and allows each relation schema in the decomposition to be in BCNF. We can check the 3NF relation schemas in the decomposition individually to see whether each satisfies BCNF. If some relation schema Ri is not in BCNF, we can choose to decompose it further or to leave it as it is in 3NF.

