Theory of Normal Forms
Decomposition of Relations

Overview

- Functional Dependencies capture the attribute dependencies within a relational table.
- Functional Dependencies that do not involve *full keys* lead to *anomalies*: problems with management/maintenance of tables.
- Normal Forms are restrictions on functional dependencies in relational tables.
- Third Normal Form (3NF) and Boyce-Codd Normal Form (BCNF eliminate from relational tables FDs that do not involve full keys[1].
- If a table in a database schema violates 3NF, often it is best to replace it with a collection of tables in 3NF. This is achieved via the procedure called **decomposition of relations**.

Relational table decomposition: replacement of a relational table $R(A_1, \ldots, A_n)$ with relational tables R_1, $R_2, \ldots, R_k$. We want relational decomopositions to have the following properties:

P1. For all i, $R_i = \pi_L(R)$, where $L \subset \{A_1, \ldots, A_n\}$.

P2. $R_1 \bowtie R_2 \bowtie \ldots R_k = R$.

Decompositions are designed to address the following issues:

- Eliination of anomalies. Property **P1** usually takes care of that.
- Recoverability of information. This is achieved by decompositions satisfying property **P2**.
- Preservation of dependencies. This is an important challenge. It may affect how far towards normalization we can get.

[1] Remember, in case of 3NF, it is possible to have an FD involving an incomplete key, but the right side of such FD hase to be a prime attribute.

Algorithms for Functional Dependencies

In order to present the algorithm for decomposing any relation into a relational schema in 3NF, we first need two algorithms involving functional dependencies:

- Closure: given a list of attributes and a set of FDs find all attributes that depend on it.
- FDs in Projection: given a relation, a collection of FDs and a list of attrbiutes, find the FDs that hold in the projection of the relation on the given attribute list.

Closure

Closure of a set of attributes. Let R be a relation, and $Q = \{A_1, \ldots, A_n\}$ be a subset of R's attributes. The closure of Q, denoted Q^+ is a set of attributes, which are functionally determined by Q, given a set S of functional dependencies on R.

FD Closure algorithm. The following algorithm computes the closure of a set of attributes.

```
Algorithm FD-Closure(R Relational_Schema, Q Set_of_Attributes, S Set_of_FDs)
     returns Set_of_Attributes;

// Step 1. split all rules in S
for each f in S
 begin
    if  (right side of f has multiple attributes)
    then replace f in S with FDs that have only on attribute on the right;
 end;

// Step 2. Include set Q in the closure

X := Q;

// Step 3. Keep including new attributes for as long as FDs allow

repeat
  X' := X;
  for each f: B1,...,Bn -> C in S
  begin
     if {B1,...,Bn} is a subset of X
     then  X := X union {C};
  end;
until (X' == X);  // repeat until no new attributes can be added to X

return X;
```

Finding Functional Dependencies in Projection

```
Algorithm FD-Project(R Relational_Schema, R1 Relational_Schema,
                     L Set_of_Attributes, S Set_of_FDs)
        returns Set_of_Attributes;
// R: original relation
// R1: projection of R onto attributes L
// S : FDs asserted on R

// initialization

   T := {};   // T is a set of functional dependencies

// main loop

   for each subset X of L
      begin
        Y := FD-Closure(R, X, S);  // for each subset of L, find its closure
        for each A in Y-X
            if A in L then T := T Union {X -> A};
      end;  // for

 return T;
```

Note: FD-Project checks *every subset* of the input set of attributes. This means, that in general case, FD-Project is exponential in the size of input. However, in practice, if the list of FDs is short, this algorithm will work faster.

Decomposition of Relations

BCNF Decomposition

The **BCNF Decomposition** algorithm, given a relation and a collection of FDs asserted on it, decomposes the relation into a number of relational tables in BCNF.

```
Algorithm BCNF-Decompose(R Relational_Schema, S Set_of_FDs)
        returns Set_of_Relational_Schema;


   if R is in BCNF then return {R}; // trivial case
   Z:= all attributes of R;

   W := {}; // initialize eventual result

// main case: R is not in BCNF, needs to be decomposed

    Find rule X-> A in S which violates BCNF;

    Y1 := FD-Closure(R,X);  // closure of left side of violating rule

    Create relational schema T1(Y1);
    S1 := FD-Project(R,T1,Y1,S);   // find FDs in the new table

    Y2 := (T - Y) Union X;
    Create relational schema T2(Y2);
    S2 := FD-Project(R,T2,Y2,S);   // find FDs in the new table

    // recursively call BCN-Decomposition algorithm
    W := W Union BCNF-Decompose(T1,S1) Union BCNF-Decompose(T2,S2);

 return W;
```

Note: BCNF-Decompose is a recursive algorithm. The idea is simple: find a rule that violates BCNF, use it to decompose the table into two. Then, find the proper set of FDs for each of the new tables, and decompose each of them in turn.

Properties of BCNF Decomposition

- Elimination of anomalies. Anomalies are eliminated, since the decomposition process *breaks off* any BCNF-violating FD into a separate table.
- Recoverability of information. Natural join **will recover** the original relation from the decomposition. Chase test allows us to verify that.
- Preservation of dependencies. This is *not always the case.* If a table was in 3NF but not in BCNF, some dependencies may be lost.

3NF Decomposition

BCNF Decomposition may lead to breaking off of FDs $X \to A$ where A is a prime attribute, i.e., part of another key. This, in turn, may yield *loss of dependency*: it would be possible to store information in the set of decomposed tables, which cannot be stored in the original table.

3NF decomposition shall be used when such FDs are present in a table, instead of BCNF decomposition. While 3NF tables are not as normalized, **they are guaranteed to always preserve all dependencies** from the original table.

Basis and Minimal Basis of FD Sets

FD Basis. Let R be a table and S be a set of FDs asserted on it. Let S' be another set of FDs, such that $S \equiv S'$. Then S' is called the basis of S.

Minimal Basis. A minimal basis for a set of FDs S is a set of FDs S', such that:

1. S' is a basis of S.
2. All FDs in S' are of the form $X \rightarrow A$, where A is a single attribute.
3. For any $X \rightarrow A \in S'$, $S' - \{X \rightarrow A\}$ is not a basis of S.
4. For any $X \rightarrow A \in S'$, $Y \subset X$, $S' - \{X \rightarrow A\} \cup \{Y \rightarrow A\}$ is not a basis of S.

Note: Informally, a minimal basis, is a basis which has only rules with one attribute on the right, and from which no rule can be removed, or simplified.

```
Algorithm 3NF-Decompose(R Relational_Schema, S Set_of_FDs)
        returns Set_of_Relational_Schema;


   if R is in 3NF then return {R}; // trivial case

   W:= {};

   G := Minimal_Basis(S);  // find minimal basis of S

   for each X->A in G
      begin
       create schema Rxa(X,A);
       W:= W Union {Rxa};
      end;

   // make certain the key of R is preserved
   if no relation from W is a superkey for R then
     begin
      Y := some key of R;
      create schema T(Y);
      W:= W Union {T};
    end;

  return W;
```

Informally, 3NF-Decompose works as follows: for each rule in the minimal basis it creates a new relation, with the left side of the rule as the key. Then, if no relation contained a key of the original table, one more relation is used to preserve the key.

Things to note about 3NF-Decompose:

- 3NF-Decompose takes care of **both** FDs that violate 3NF and FDs that violate 2NF.
- 3NF-Decompose may split the table into more relations than necessary. If there are two FDs: $X \rightarrow A$ and $X \rightarrow B$ in the minimal basis, then two tables $Rxa(X, A)$ and $Rxb(X, B)$ will be created. At the same time, a single table $Rx(X, A, B)$ is sufficient in many cases (if both A and B are non-prime).

Properties of 3NF-Decompose :

- Lossless join. Chase procedure can be used to verify that 3NF-Decompose yields lossless join.

- Dependency preservation. **Every FD** from the minimal basis **is preserved in one table**!
- 3NF. If any of the relations Rxa created during decomposition is NOT in 3NF, then, the basis we used during the decomposition is not minimal.