Design Theory for Relational Databases

Based on the Chapter in “The Complete Database Book”
by Garcia-Molina, Ullman and Widom

CS315: Principles of Database Systems, Jan-Apr 2022
IIT Kanpur

Outline

Functional Dependencies: Introduction

- The design theory presents a way of formulating constraints that apply to a relation.
- **Functional Dependencies** are a common type of constraint.
- Generalizes the idea of a *key* of a relation.
- Gives tools to improve our designs by the process of "*decomposition*" of relations.

Definition of FD

- Functional dependency is a statement of constraint on a relation schema R.
- Suppose there are attributes $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$, all belonging to R.
- A functional dependency is denoted as

$$A_1, A_2, \ldots, A_n \to B_1, B_2, \ldots, B_m$$

read as

$$A_1, A_2, \ldots, A_n \text{ functionally determine } B_1, B_2 \ldots, B_m \quad .$$

- It means: If any two tuples of R agree on all their respective attribute values $A_1, A_2, \ldots, A_n$, then, they must also agree on all of the attribute values $B_1, B_2, \ldots, B_m$.

Definition of FD: contd.

- Formally, for any two tuples $s, t \in r(R)$, if

  $s[A_1, A_2, \ldots, A_n] = t[A_1, A_2, \ldots, A_n]$ then $s[B_1, \ldots, B_m] = t[B_1, \ldots, B_m$

- An FD is specified as a constraint *if every database instance r over the schema R will satisfy this FD*.
- When we say the schema R satisfies an FD, we are asserting a constraint for all valid instances of relation r over R.
- The FD $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$ is equivalent to the set of m FDs

$$\begin{array}{ll} A_1, \ldots, A_n & \rightarrow B_1 \\ A_1, \ldots, A_n & \rightarrow B_2 \\ \vdots & \vdots \\ A_1, \ldots, A_n & \rightarrow B_m \end{array}$$

A simple running example

- Consider the relation
  `Movies1(title, length, genre, studioName, starName, starDoB, starGender)`
  with extends the `Movies` relation earlier by adding `studioName` and `starName`.
  1. `starName` refers to the names of stars who have played a role in this movie.
  2. `studioName` refers to the name of the production studio for this movie (there is only one!)
- Following FD holds

$$\texttt{title} \rightarrow \texttt{length, genre, studioName}$$

meaning, any two tuples with the same title (i.e., same movie) have the same `length`, `genre` and `studioName`.

- Note that the statement

$$\texttt{title} \rightarrow \texttt{starName}$$

  is not correct; it is not a *functional dependency*.
- Given a movie, it may well have multiple stars playing the same/different roles.

Keys

- A set of one or more attributes $\{A_1, A_2, \ldots, A_n\}$ is a key for a relation schema R if
- These attributes functionally determine all other attributes of the relation.
  - Thus, it is impossible for two distinct tuples of R to agree on all of $A_1, \ldots, A_n$–all key attributes.
- No proper subset of $\{A_1, \ldots, A_n\}$ functionally determines all the other attributes of R.
- E.g., `title, starName` forms a key for the schema `Movies1`.
- No proper subset of `title, starName` determines all other attributes; hence it is a key.

Keys

- Subsets such as `genre, starName, studioName` is not a key.
    - There can be multiple movies that are produced in the same `studioName` with role for the same `starName` and of same `genre`.
    - Thus,

      $$\texttt{genre, starName, studioName} \to \texttt{title}$$

      is not an FD that will be satisfied by all legal relations under schema `Movies1`.
- Sometimes a relation may have more than one key.
    - Common to designate one of the keys as the *primary key*.
    - Valid in SQL.

Superkey

- A set of attributes that contains a key is called a *superkey*.
- E.g., {`title, starName, genre`} is a superkey for `Movies1`.
- Any superset of {`title, starName`} is a superkey.

Reasoning about functional dependencies: Example

- Consider relation schema $R(A, B, C)$ that satisfies the FDs

$$A \to B, \quad B \to C \ .$$

- We should be able to deduce that $A \to C$.
  - Take any two distinct tuples s, t from any relation $r(R)$ satisfying these FDs.
  - Suppose they agree on A: $s[A] = t[A]$.
  - Since $A \to B$, then, $s[B] = t[B]$.
  - Since $B \to C$, hence, $s[C] = t[C]$.
  - In other words, the FD $A \to C$ holds.
- This is the *transitive rule*.

Reasoning about FDs: another notation

- Suppose $A \rightarrow B$ and $B \rightarrow C$ holds over $R(A, B, C)$.
- Let the tuples agreeing on attribute a be (a, b_1, c_1) and (a, b_2, c_2).
- Since the FD $A \rightarrow B$ holds, and the tuples agree on A, they must also agree on B.
- Hence $b_1 = b_2$.
- Since $B \rightarrow C$ holds, the tuples now agreeing on b must agree on C. Thus, $c_1 = c_2$.
- Hence, the FD $A \rightarrow C$ holds.

Equivalence of FD sets

- Sets of FDs can be written in different, equivalent ways.
- Two sets of FDs S and T are *equivalent* if whenever there is a relation instance that satisfies S, it satisfies T and vice-versa.
- A set of FDs S *follows* from a set of FDs T if every relation instance that satisfies all the FDs in T also satisfies all the FDs in S.
- Hence, set of FDs S and T are equivalent, if S follows from T and vice-versa.

Some Useful things regarding FDs

- Some useful rules:
  1. Replace one set S of FDs by an equivalent set T.
  2. Add to the set S of FDs, a set of FDs that follow from S.
  3. Remove or modify (reduce) FDs from a given set S, and yet maintain equivalence.

Splitting/Combining Rule

- *Splitting rule*: Given an FD $A_1 A_2 \ldots A_n \to B_1 B_2 \ldots B_m$, we can replace it by a set of FDs:

$$A_1 A_2 \cdots A_n \to B_i, \quad i = 1, 2, \ldots, m$$

- *Combining rule*: Given a set of FDs

$$A_1 A_2 \cdots A_n \to B_i, \quad i =, 2, \ldots, m$$

  we can combine these FDs into a single FD $A_1 A_2 \ldots A_n \to B_1 B_2 \ldots B_m$.

Splitting/Combining rule: Example

- The set of FDs

$$\begin{aligned} \texttt{title, starName} &\rightarrow \texttt{length} \\ \texttt{title, starName} &\rightarrow \texttt{genre} \\ \texttt{title, starName} &\rightarrow \texttt{studioName} \end{aligned}$$

  is equivalent to a single FD

$$\texttt{title, starName} \rightarrow \texttt{length, genre, studioName} .$$

No splitting on the *LHS*

- Consider the FD `title, starName` $\to$ `length`.
- It is not equivalent to splitting the *LHS* into two FDs

$$\begin{array}{ll} \texttt{title} & \to \texttt{length} \\ \texttt{starName} & \to \texttt{length} \end{array}$$

This set of FDs is *false*.

Trivial Functional Dependencies

- E.g., `title, starName` $\rightarrow$ `starName` is trivial.
- Any two tuples in the `Movies1` table that have the same values in the `title` and `starName` field obviously agree on the attribute value `title`.
- `title, starName`$\rightarrow$ `title` is rather obvious.
- For any subset $\{B_1, B_2, \ldots, B_m\}$ of the attribute subset $\{A_1, \ldots, A_n\}$, the FD

$$A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$$

  always holds.

Almost Trivial FDs

- E.g., {texttttitle, starName } $\rightarrow$ `genre, studioName, title`.
- Since `title` is in the LHS, it determines itself and hence it is trivial that `title` is in the *RHS*.
- The above is equivalent to

$$\texttt{title, starName} \rightarrow \texttt{genre, studioName}$$

.

- The FD $A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_m$ is equivalent to

$$A_1 A_2 \dots A_n \rightarrow C_1 C_2 \dots C_k$$

where the C's are all the B's that are not also A's.

Closure of Attribute Set

- Given a set S of FDs and a set of attributes $\{A_1, A_2, \ldots, A_n\}$.
- The *closure* of $\{A_1, A_2, \ldots, A_n\}$ under the set of FDs S:

$$\{B \mid \text{ every relation satisfying all FDs in } S \text{ also satisfies } A_1A_2\cdots A_n -$$

- Denoted as $\{A_1, A_2, \ldots, A_n\}^+$.
- Note that $A_1, \ldots, A_n$ are always in $\{A_1, \ldots, A_n\}^+$, since,
  - The FD $A_1A_2\cdots A_n \to A_i$ is *trivial* when i is in $1, 2, \ldots, n$.

Algorithm for computing closure

- INPUT: A set of attributes $\{A_1, A_2, \ldots, A_n\}$ and a set of FDs S.
- OUTPUT: The closure $\{A_1, A_2, \ldots, A_n\}^+$.

```
Split FDs of S if needed so that RHS of each FD
is a single attribute.
Initialize X = {A1, A2, ..., An}
repeat
        Find some FD B1B2 ... Bm → C in S such that
        {B1, B2, ..., Bm} ⊂ X and C ∉ X
        Add C to X:   X := X ∪ {C}.
until such an FD cannot be found.
return X
```

Remarks on the closure algorithm

- In each step, the algorithm either increases X by one attribute or it terminates.
- The number of attributes in a relational schema is finite–hence the algorithm must terminate.

Closure algorithm: Example

- Let schema be $R = (A, B, C, D, E, F)$. Set of FDs F is

$$AB \to C, \qquad BC \to AD \qquad D \to E \qquad CD \to B$$

- Find $\{A, B\}^+$. We run the closure algorithm.

1. Initialize $X = \{A, B\}$.
2. Use $AB \to C$. Add C to X. $X = \{A, B, C\}$.
3. Use $BC \to D$. Adds D to X. $X = \{A, B, C, D\}$.
4. Use $D \to E$. Adds E to X. $X = \{A, B, C, D, E\}$.
5. Algorithm terminates. $\{A, B\}^+ = \{A, B, C, D, E\}$.

Test for Implication

- Suppose we are given an FD set S and another FD $A_1 A_2 \cdots A_n \to B$.
- Question: Does S imply $A_1 A_2 \cdots A_n \to B$?
- A solution.
  1. Find the closure $\{A_1, A_2, \ldots, A_n\}^+$ using the set of FDs S.
  2. If B is in $\{A_1, A_2, \ldots, A_n\}^+$, then, S implies $A_1 A_2 \cdots A_n \to B$.
  3. Otherwise, $A_1 A_2 \cdots A_n \to B$ does not follow from S.
- Generalizing: Check if S implies $A_1 A_2 \cdots A_n \to B_1 B_2 \cdots B_m$.
- Solution: Implication is iff provided

$$B_1, B_2, \ldots, B_m \text{ are all in } \{A_1, A_2, \ldots, A_n\}^+ \quad .$$

Example

- $R = (A, B, C, D, E, F)$. Set S of FDs is

$$AB \to C, \qquad BC \to AD \qquad D \to E \qquad CD \to B$$

- Test: Does $AB \to D$ follow from S?
  1. $\{A, B\}^+ = \{A, B, C, D, E\}$ (previous calculation).
  2. D is in $\{A, B\}^+$, hence, $AB \to D$ follows.
- Test2: Does $D \to A$ follow? First let us compute $\{D\}^+$.
  1. $X = \{D\}$.
  2. Using $D \to E$, we get $X = \{D, E\}$.
  3. We now terminate. $\{D\}^+ = \{D, E\}$.
- $A \notin \{D\}^+$. So $D \to A$ does not follow from S.

Why does the closure algorithm work

- Summary: the closure algorithm takes a set S of FDs and a set $\{A_1, A_2, \ldots, A_n\}$ and returns the closure set $\{A_1, A_2, \ldots, A_n\}^+$ such that for each B in the closure set S
  S implies that $A_1 A_2 \ldots A_n \rightarrow B$.
- For correctness of the closure algorithm, we show two properties.
  1. If $A_1 A_2 \ldots A_n \rightarrow B$ is implied by the closure algorithm, then, $A_1 A_2 \cdots A_n \rightarrow B$ holds in each relation instance that satisfies S. (*Soundness*)
  2. The closure algorithm does not fail to discover any true FD $A_1 A_2 \cdots A_n \rightarrow B$ that truly follows from S. (*Completeness*).

Proof of part: doesn't claim too much (Soundness)

- Proof idea:
  1. Initial step $X = \{A_1, A_2, \ldots, A_n\}^+$ obviously is correct: trivial FDs.
  2. In successive step, we would like to show that for every C in X, the FD $A_1 A_2 \cdots A_n \to C$ is implied from S.
  3. In the first step of the algorithm, say we consider an FD $B_1 B_2 \cdots B_n \to C$.
     1. Each of the B_i's is in X and C is not in X. Hence, we add C to X.
     2. Consider any relation instance that satisfies all FDs in S.
     3. Suppose there are two tuples that agree on the attributes $A_1, A_2, \ldots, A_n$.
     4. From the FD $B_1 B_2 \cdots B_m \to D$ of S, each of B_i's is in X, which is now $\{A_1, A_2, \ldots, A_n\}$.
     5. The two tuples agree on each of the B_j's.
     6. Hence from $B_1 B_2 \cdots B_m \to C$, the two tuples agree on C.
     7. So the instance satisfies $A_1 A_2 \cdots A_n \to C$.

Proof of Soundness

1. Proof uses induction on the number of times k the set X grew by one item at a time.
2. *Base case:* Zero steps. Correctness follows from trivial FDs: $X = \{A_1, A_2, \ldots, A_n\}$.
3. *Induction hypothesis:* After the $k-1$th iteration, for $k \geq 1$, we assume that for every D in X, the FD $A_1 A_2 \cdots A_n \rightarrow D$ is implied by S.
4. Consider the kth iteration where an item C was added by the algorithm using the FD $B_1 B_2 \cdots B_m \rightarrow C$ and each of the B_i's is in X.

Proof: Soundness-II

6. Consider any instance of the relation that satisfies all the FDs in S.
7. Suppose there are two tuples that agree on all the attributes of $A_1, A_2, \ldots, A_n$.
8. By the induction hypothesis, the FD $A_1A_2 \cdots A_n \rightarrow B_i$, for each $i = 1, \ldots, m$, holds in S.
   - In particular $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ holds (by combining).
9. The two tuples agree on the values of each of the B_i's, $i = 1, \ldots, m$.
10. From the FD $B_1 \ldots B_m \rightarrow C$ in S, the two tuples also agree on attribute C.
11. Hence, $A_1A_2 \ldots A_n \rightarrow C$ holds. (Proof by induction).

Proof: Algorithm did not miss out any true FD (Completeness)

- Suppose $A_1 A_2 \cdots A_n \to B$ is an FD that is implied by S but the closure algorithm failed to include $B \in \{A_1, A_2 \ldots, A_n\}^+$.
- We will construct a relation instance I such that
  1. I satisfies all the FDs in S, but
  2. I does not satisfy $A_1 A_2 \cdots A_n \to B$.

  This implies that $A_1 A_2 \cdots A_n \to B$ is not implied by S.

Constructing the counter-example instance

- Construct a relation instance with two tuples t and s as follows.
- It has all the attributes of the original schema R, divided into two parts: $\{A_1, A_2, \ldots, A_n\}^+$ and $R - \{A_1, A_2, \ldots, A_n\}^+$.

	$\{A_1, A_2, \ldots, A_n\}^+$	Other Attributes
t:	1 1 $\cdots$ 1 1	0 0 $\cdots$ 0 0
s:	1 1 $\cdots$ 1 1	1 1 $\cdots$ 1 1

- Does it satisfy all FDs in S?
- Suppose there is $C_1 C_2 \ldots C_k \rightarrow D$ in S that I doesn't satisfy.

instance I:

	$\{A_1, A_2, \ldots, A_n\}^+$	Other Attributes
t:	$1\ 1\ \cdots 1\ 1$	$0\ 0\ \cdots 0\ 0$
s:	$1\ 1\ \cdots 1\ 1$	$1\ 1\ \cdots 1\ 1$

1. There are only two tuples in I, t and s.
   - If in this instance, the FD $C_1 C_2 \ldots C_k \rightarrow D$ is not satisfied, then,
   - they must agree on $C_1, C_2, \ldots, C_k$ attributes and disagree on D.
2. The only way they can agree on instance I is if each of the C_i's is in $\{A_1, \cdots, A_n\}^+$, since,
   - any C_i belonging to the *Other Attributes* will disagree.
3. However, if $\{C_1, \ldots, C_k\} \subset \{A_1, \cdots, A_n\}^+ = X$ (final iteration),
   - using FD $C_1 \cdots C_k \rightarrow D$ from S, the closure algorithm would include D in $\{A_1, \cdots A_n\}^+$.
   - i.e., the algorithm could not have terminated without including D.

Counter-example instance

In the counter example instance I:

1. There is no FD in S that is not satisfied, OR, all FDs in S are satisfied in I.
2. The proposed FD $A_1 A_2 \cdots A_n \rightarrow B$, where, $B \notin \{A_1, A_2, \ldots, A_n\}^+$ is not satisfied. Why?
    1. In the instance, t and s have 1's in all the attributes in $\{A_1, A_2, \ldots, A_n\}^+$, but differ in each of the other attributes as 0/1.
    2. Since, $B \notin \{A_1, A_2, \ldots, A_n\}^+$, the FD $A_1 A_2 \cdots A_n \rightarrow B$ does not hold in I.
3. This means that $A_1 A_2 \cdots A_n \rightarrow B$ is not implied by the FD set S.
4. *Conclusion*: The closure algorithm did not miss any FD implied by S. *Proved*.

Transitive Rule

- *Transitive Rule:* If $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \rightarrow B_m$ and $B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k$, then $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$ also holds in R.
- To see this: take the closure test.
  1. What is $\{A_1, A_2, \ldots, A_n\}^+$? Among other attributes it includes $\{B_1, \cdots, B_m, C_1, \cdots, C_k\}$ (using closure algorithm).
  2. Hence, $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$.

Closures and Keys

- $\{A_1, A_2, \ldots, A_n\}^+$ is a superkey of a relation R if and only if $\{A_1, A_2, \ldots, A_n\}^+ = R$, that is, the closure of $A_1, \ldots, A_n$ includes all the attributes of R.
- Because, only then can $\{A_1, \cdots, A_n\}$ functionally determine all the other attributes.
- Test if $A_1, A_2, \ldots, A_n$ is a key?
  1. $\{A_1, A_2, \ldots, A_n\}^+$ is the set of all the attributes of R.
  2. For every subset X obtained by removing one attribute say A_i from $\{A_1, A_2, \ldots, A_n\}^+$, X^+ does not include all attributes of R.
     - i.e., X^+ does not include A_i.

Basis (Cover) of FDs

- Suppose we are given a set of FDs *S* for relations over a schema *R*.
- Any set of FDs that are equivalent to *S* is said to be the basis for *S*.
- *Notation.* Assume that all FDs have singleton *RHS*.
- *Minimal Basis* for *S* is a basis *B* that satisfies these conditions.
  1. All FDs in *B* have singleton *RHS*.
  2. If any FD is removed from *B*, it is no longer a basis for *S*.
  3. If for any FD in *B*, we remove one attribute from the *LHS* of any FD in *B*, it is no longer a basis for *S*.

Example of Minimal Basis

- Consider a schema $R(A, B, C)$ such that each attribute functionally determines the other two.
- It can be written as

$$A \to B \qquad B \to C \qquad C \to A$$
$$A \to C \qquad B \to A \qquad C \to B$$

- Minimal Base 1:

$$A \to B \qquad B \to A \qquad B \to C \qquad C \to B$$

Also minimal base 2:

$$A \to B \qquad B \to C \qquad C \to A$$

Finding a minimal basis

- *Problem*: Given a schema R and a set of FDs S, find a minimal basis T of S.
- Use splitting of the *RHS* so that each of the FDs in S has a singleton attribute.
- The problem is solved in two phases.
- *Phase 1:* Remove dependencies in S that are implied by all the others in S.
  - At the end of phase 1, we get a set of dependencies say S', s.t.
  - it is S minus some of its dependencies deleted.
  - S and S' are equivalent, that is, S' is a basis of S.
  - No FD in S' can be removed while preserving equivalence.

Finding minimal basis

- *Phase 2: Input* is the set S' obtained after phase 1.
- *Output* is a set of dependencies T that is equivalent to S' (hence equivalent to S). T is a basis for S.
  - *LHS* of each of the dependencies in T is minimal, that is,
  - Removing any attribute from the*LHS* of any FD in T does not preserve equivalence with T.
- In both phases, the final output is dependent on
  1. the ordering of the FDs (phase 1), and
  2. ordering of attributes in the *LHS* of each of the FDs.
- Minimal basis of a set S of FDs may not be unique.

Finding a minimal basis: Phase 1

1. Place the FDs in S in some order.
2. For each FD F in S (in order) check whether it follows from other FDs in S.
   1. Let $F = A_1 A_2 \cdots A_n \rightarrow B$.
   2. Let $S' = S - \{F\}$ (remove F from S).
   3. Compute the closure $\{A_1, A_2, \ldots, A_n\}^+$ under S'.
   4. If B is in $\{A_1, \ldots, A_n\}^+$ remove F from S: S **:=** $S - \{F\}$,
   5. else continue to the next FD of S in the given order.
3.
   1. Output set S is equivalent to the original set of FDs S.
   2. No FD in output S is redundant– for every $F \in S$, $S - \{F\}$ is not equivalent to S.

Phase 2

- *Input* is the set of FDs S which is the output of phase 1.
  1. Order the FDs in the input in some order.
  2. Order all the attributes of R in some order.
  3. For every FD $A_1 A_2 \cdots A_n \rightarrow B$, the attributes in the *LHS* are placed in the global attribute order.

Phase 2 algorithm

for every FD F in S in given order with $LHS \geq 2$ attributes
    let F be $A_1 A_2 \cdots A_n \rightarrow B$, $n \geq 2$
    let LHS **:=** $\{A_1, A_2, \ldots, A_n\}$
    for each attribute A_i in succession, $i = 1, 2, \ldots, n$
        // Remove A_i from LHS to obtain LHS'
        let LHS' **:=** $LHS - \{A_i\}$
        Take closure of LHS' under S and
            check if LHS'^{+} includes B.
        if TRUE
            // Replace F in S by removing A_i from its LHS
            replace F in S by F **:=** $A_1 \cdots A_{i-1} A_{i+1} \cdots A_n \rightarrow B$.
            LHS **:=** LHS' //update LHS
return S

Remarks 1

- Let S be the given set of FDs and F be any FD in S.
- If G is any functional dependency that is logically implied by $S - \{F\}$, then it is also logically implied by S. *Why?*
- Let G be $B_1 B_2 \cdots B_m \rightarrow C$.
- If G is logically implied by $S - \{F\}$, then, C is in $\{B_1, B_2, \ldots, B_m\}^+$ where the closure is taken under $S - \{F\}$.
- Clearly, the closure of $\{B_1, B_2, \ldots, B_m\}^+$ under S contains all the items in the closure of $\{B_1, B_2, \ldots, B_m\}^+$ under $S - \{F\}$.
- So C is in the closure of $\{B_1, B_2, \ldots, B_m\}^+$ under S.

Remarks 1a

- Let S be a set of FDs that includes F: $A_1A_2 \cdots A_n \rightarrow B$.
- The test for whether S is equivalent to $S - \{F\}$ is equivalent to checking if F is implied by $S - \{F\}$.
- Or, the closure $\{A_1, A_2, \ldots, A_n\}^+$ under $S - \{F\}$ includes B.

Remarks 2

- Let S be a set of FDs that includes F: $A_1 A_2 \cdots A_n \rightarrow B$.
- Let $X = \{A_1, A_2, \ldots, A_n\}$ and $Y = X - \{A_i\}$.
- Consider the set of FDs

$$S' = (S - \{F\}) \cup \{Y \rightarrow B\} \ .$$

- Which of the sets of FDs S or S' is "stronger"? i.e., which one implies the other?
- S' is stronger, i.e., S' logically implies S. Why?
- Clearly, each FD F' in S that is not equal to F is in S'.
- S' implies F; since,
- under S', $X^+ \supset Y^+$ and Y^+ contains B and so $F : X \rightarrow B$ is implied by S'.

Remark 2a

- Following prior notation, given set of FDs S with $F : X \to B$ and $F' : Y \to B$, where $X = \{A_1, A_2, \ldots, A_n\}$ and $Y = X - \{A_i\}$.
- $S' = (S - \{F\}) \cup \{Y \to B\}$.
- From previous discussion, S and S' are equivalent, iff $Y \to B$ is implied by S.
- To check this, take the closure Y^+ under S and check if B is in the closure.
- If yes, then they are equivalent, and if no, then $Y \to B$ is not implied by S,

Projecting Functional Dependencies

- We are given a relation R and set of FDs S.
- Suppose we project R onto a subset L of attributes

$$R_1 = \pi_L(R) \ .$$

- Question: What are the FDs implied by S that hold on R_1?
- *This is called the projection of functional dependencies of S on* R_1, that is,
- All FDs that follow from S and involve only attributes of R_1.
- Projection of S on a projection of R onto R_1 is a basic problem.

Projecting a set S of Functional Dependencies

1. INPUT: (1) A relation R and $R_1 = \pi_L(R)$. (2) A set of FDs S that hold in R.
2. OUTPUT: The set of FDs that hold in R_1.
3. Let T be the eventual output set of FDs. Initialize T to empty set.
4. For each subset X of attributes of L (schema of R_1), compute X^+ under S.
5. Add to T all non-trivial FDs

$$X \to A, \quad \text{where, } A \text{ is in } X^+ \text{and in } R_1 \quad .$$

6. Construct a minimal basis for T and return it.

Projection of a set of FDs: Example

- Suppose $R(A, B, C, D)$ has FDs

$$A \to B \qquad B \to C \qquad C \to D$$

- We wish to project out B giving $R_1(A, C, D)$. Find the projections of the FD set on R_1 schema (A, C, D).
- Start with singletons and compute their closure under the given FDs.
  1. $\{A\}^+ = \{A, B, C, D\}$. Hence on R_1 the following FDs hold:

$$A \to C \qquad A \to D \; .$$

  2. $\{C\}^+ = \{C, D\}$. In R_1, $C \to D$ holds.
  3. $\{D\}^+ = \{D\}$. No more FDs are added.
- From the singleton LHS, we get the following FDs on R_1:

$$A \to C \qquad A \to D \qquad C \to D \; .$$

Example

- We should now consider doubletons.
- Since $\{A\}^+$ includes all attributes of R, there is no point considering supersets of A, such as AC or AD or ACD. Under R_1, $\{A, C\}^+ = \{A, D\}^+ = \{A, C, D\}$.
- Also, $\{C, D\}^+ = \{C, D\}$, giving only the trivial dependency.
- So the set of FDs on the projection $R_1(A, C, D)$ are:

$$A \to C \qquad A \to D \qquad C \to D \ .$$

- In the above set, the FD $A \to D$ is redundant; by transitivity $A \to C$ and $C \to D$ gives $A \to D$.
- Removing, $A \to D$, we are left with a minimal basis for the projection of FDs on R_1:

$$A \to C \qquad C \to D \ .$$

Design of Relational Database Schemas: Overview

- Consider the "long" schema
  `Movies1(title, length, genre, starName, starDoB, role)`
- This schema has several issues: *Redundancy* or *Repetition of Information*, *Update Anomaly*, *Deletion Anomaly*.
- We introduce the idea of **decomposition**; breaking a relation schema into two (or more) smaller schemas.
- Next, we introduce the Boyce-Codd normal form or **BCNF**: a condition on the relation schema that eliminates these problems.
- Finally, we give a method to decompose relation schemas to satisfy the **BCNF** condition.

Anomalies

- *Redundancy*. The same information about an entity is repeated in several tuples.

title	*length*	*genre*	*starName*	*starDoB*	*role*
Baazigar	182	Thriller	Shahrukh Khan	1965-11-02	Male Actor
Baazigar	182	Thriller	Kajol	1974-08-05	Female Actor
Baazigar	182	Thriller	Anu Malik	1960-11-02	Music Director
Chak De! India	153	Sports	Shahrukh Khan	1965-11-02	Male Actor
Lagaan	224	Drama	Amir Khan	1965-03-14	Male Actor

- For “Baazigar” film, the information about length and genre is repeated several times; the information about actor “Shahrukh Khan” is repeated several times for several movies, etc..

Update Anomaly

- For example, suppose we wish to update the length of movie 'Baazigar' to 185 minutes.
- We may make the mistake of updating the length attribute in one tuple with 'Baazigar' but leave it unchanged in another tuple with 'Baazigar'.
- This would make the data inconsistent.
- Or, one may argue that to maintain consistency, we have to update all tuples having the movie title 'Baazigar'.
- This is a bit expensive; but it is possible to redesign the `Movies1` relation so that such issues do not arise.

Delete Anomaly

- If a set of values becomes empty then we may lose other information as a side effect.
- For e.g., if we delete 'Amir Khan' from the stars of the movie titled 'Lagaan', we would additionally lose information about the genre and length of the movie 'Lagaan'.

Decomposing Relations

- An accepted way to eliminate these anomalies is to *decompose* relations.
- *Basic Decomposition Step*: Partition the schema of R into two schemas with overlapping attributes and unique attribute(s).
- Given relation $R(A_1, A_2, \ldots, A_n)$, we *decompose* into two relations $S(B_1, B_2, \ldots, B_m)$ and $T(C_1, C_2, \ldots, C_k)$ such that:

  1. $\{A_1, A_2, \ldots, A_n\} = \{B_1, B_2, \ldots, B_m\} \cup \{C_1, C_2, \ldots, C_k\}$
  2. $S = \pi_{B_1, B_2, \ldots, B_m}(R)$
  3. $T = \pi_{C_1, C_2, \ldots, C_k}(R)$

Decomposition: Example

- Consider the `Movies1` relation.
  `Movies1(title, length, genre, starName, starDoB, role)`
- Suppose we decompose it into two relations, called `Movies` and `StarsIn` with the following schema:

  1. `Movies(title, length, genre)`
  2. `StarsIn(title, starName, starDoB, role)`

- Here,

$$\texttt{Movies} = \pi_{\texttt{title, length, genre}}(\texttt{Movies1})$$
$$\texttt{StarsIn} = \pi_{\texttt{title, starName, starDoB, role}}(\texttt{Movies1})$$

Example

title	length	genre
Baazigar	182	Thriller
Chak De! India	153	Sports
Lagaan	224	Drama

Figure: The relation Movies

title	starName	starDoB	role
Baazigar	Shahrukh Khan	1965-11-02	Male Actor
Baazigar	Kajol	1974-08-05	Female Actor
Baazigar	Anu Malik	1960-11-02	Music Director
Chak De!India	Shahrukh Khan	1965-11-02	Male Actor
Lagaan	Amir Khan	1965-03-14	Male Actor

Figure: The table StarsIn

Advantage of Decomposition

The following are some of the advantages obtained decomposing `Movies1` into `Movies` and `StarsIn`

- The redundancy regarding each tuple in `Movies` is eliminated: the basic record regarding each film appears only once.
- The update anomaly is gone: suppose we change the length of 'Baazigar' to 185, then we just change it in only one record in the `Movies` relation.
- The delete anomaly is gone: if for some reason, we delete all the stars from the movie 'Baazigar', the information regarding the movie 'Baazigar' still remains in the table `Movies`.

- There is no redundancy regarding multiple occurrences of `title` in relation `StarsIn`
  The `title` attribute is a key for `Movies` table and a movie can appear several times in the `StarsIn` table.
  The `title` attribute represents a movie succintly.

Boyce-Codd Normal Form

- Goal of decomposition: replace a relation by several so that no anomalies exist.
- **BCNF** gives a simple condition under which the anomalies discussed above will not exist.
- A relation R is in *Boyce-Codd normal form* or **BCNF** if and only if: *for every non-trivial FD* $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ *that holds for* R, $\{A_1, A_2, \ldots, A_n\}$ *is a superkey for* R.
- That is, the *LHS of every non-trivial dependency is a superkey for R.*
- Recall that a superkey need not be minimal.
  Equivalently, BCNF means that the *LHS* of every FD contains a key.

Example contd.

- Let's apply the BCNF condition to the relation `Movies1` `Movies1(title, length, genre, starName, starDoB, role)`.
- `Movies1` is not in BCNF because we have the FD

$$\texttt{title} \to \texttt{length, genre}\ .$$

- But `title` is not a superkey. Hence it is not in BCNF.
- We also have the FD

$$\texttt{starName} \to \texttt{starDoB}$$

- Finally, assuming a star can play multiple roles in the same movie, `role` is not functionally determined by `title` and `starName`.
- Key for `Movies1` is `title, starName, role`.

Example

- The decomposition of `Movies1` into

  `Movies(title, length, genre)` and
  `StarsIn(title, starName, starDoB, role)`

  is slightly better.
- The relation `Movies(title, length, genre)` is now in BCNF. The only FD is

  $$\texttt{title} \rightarrow \texttt{length, genre}$$

- So {`title`} is a key and `Movies` is a superkey.
- Note that it is the only key of this relation.

Example

- However, the relation `StarsIn(title, starName, starDoB, role)` is not in BCNF. Let's see why?
- The FDs are:

$$\texttt{starName} \rightarrow \texttt{starDoB}$$

  There are no other FDs.
- The only key is {`title, starName, role`}.
- Hence, `StarsIn` is not in BCNF.

Decomposition into BCNF

- By suitably choosing decompositions, we can decompose any relation schema into a collection of subsets of its attributes with the following properties.

1. Each subset in the decomposition is a schema of relations in BCNF.
2. The decomposition is *lossless*, that is, by taking a natural join of all the projections of the original relation onto decomposition subsets, we reconstruct the original relation exactly.

- After the decomposition, the original relation data can be faithfully reconstructed by a natural join of decomposed relation instances, each of which is a projection of the original relation R onto the decomposition subset attributes.

BCNF decomposition

- Let $R(A_1, A_2, \ldots, A_n)$ be the relation R with its schema.
- *Rule:* Suppose there is a non-trivial dependency for R

$$B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k$$

  1. Each of B_i's and C_j's are attributes from the schema of R.
  2. None of the C_j's appears among the B_i's (fully non-trivial *RHS*).
- Decompose R into two relations R_1 and R_2 with the following schema:

Schema of $R2$: $\{B_1, B_2, \ldots, B_m, C_1, C_2, \ldots, C_k\}$
schema with attributes on both sides of the FD.

Schema of $R1$: $\{A_1, A_2, \ldots, A_n\} - \{C_1, C_2, \ldots, C_k\}$
original schema except all attributes in the *RHS*.

Example

- We revisit the relation `Movies1` with schema: `Movies1(title, length, genre, starName, starDoB, role)`.
- Consider the FD

$$\texttt{title} \rightarrow \texttt{length, genre}$$

.

- The BCNF rule decomposes Movies1 into two relations

`Movies12(title, length, genre)`

`Movies11(title, starName, starDoB, role)`

- The schema `Movies12(title, length, genre)` consists of all the attributes on either side of the FD.
- The schema `Movies11(title, starName, starDoB, role)` consists of all attributes of `Movies` except for the attributes in the *RHS* of the FD.

BCNF decomposition: One step

- In the earlier example, we named
  1. `Movies12` as `Movies`, and
  2. `Movies11` as `StarsIn`.
- A decomposed relation is a projection on the attributes of its schema subset.

$$\texttt{Movies12} = \pi_{\texttt{title, length, genre}}(\texttt{Movies1})$$

$$\texttt{Movies11} = \pi_{\texttt{title, starName, starDoB, role}}(\texttt{Movies1})$$

- In `Movies12`, there is only one FD: $\texttt{title} \rightarrow \texttt{length, genre}$. Hence, it *is* in BCNF.
- Is

$$\texttt{Movies11} \bowtie \texttt{Movies12} = \texttt{Movies?}$$

This is true and is called a *lossless join decomposition*. We will soon get to this.

BCNF decomposition: Example

- Consider `Movies11(title, starName, starDoB, role)`. Is it in BCNF?
- Suppose we assume that it is possible for a star to play multiple roles. So the FD

  $$\texttt{title, starName} \rightarrow \texttt{role} \text{ is FALSE.}$$

- We have the FD on `Movies11`:

  $$\texttt{starName} \rightarrow \texttt{starDoB}$$

  and `starName` is clearly not a key/superkey of `Movies11`.
- Following *BCNF decomposition rule*, `Movies11` decomposes into

  `Movies112(starName, starDoB)`

  `Movies111(title, starName, role) .`

- *Conclusion*: Both `Movies112` and `Movies111` are in BCNF.

BCNF decomposition: Example

- Applying BCNF decomposition rule twice gives the following three relation schemas.

$$\texttt{Movies12(title, length, genre)}$$
$$\texttt{Movies112(starName, starDoB)}$$
$$\texttt{Movies111(title, starName, role)}$$

- Each decomposition is a projection of `Movies` into its subset schema attributes.

$$\texttt{Movies111} = \pi_{\texttt{title, starName, role}}(\texttt{Movies1})$$
$$\texttt{Movies112} = \pi_{\texttt{starName, starDoB}}(\texttt{Movies1})$$
$$\texttt{Movies12} = \pi_{\texttt{title, length, genre}}(\texttt{Movies1}) \ .$$

BCNF decomposition: Example

- We remarked that the decomposition of `Movies1` into `Movies12` and `Movies11` was *lossless*:

$$\texttt{Movies1} = \texttt{Movies11} \bowtie \texttt{Movies12} \ .$$

- Equally, the second decomposition of `Movies11` into `Movies112` and `Movies111` is also *lossless*:

$$\texttt{Movies11} = \texttt{Movies111} \bowtie \texttt{Movies112} \ .$$

- Therefore, the two step decomposition into three fragments is *lossless*, i.e.,

$$\texttt{Movies1} = \texttt{Movies12} \bowtie \texttt{Movies111} \bowtie \texttt{Movies112}$$

BCNF decomposition

- In general, we keep applying the BCNF decomposition rule as many times as needed, until all relations (fragments) are in BCNF.
- Each time we apply the decomposition rule, each of the two resulting fragment schemas each have fewer attributes than the starting schema.
- The process must terminate. (*Note:* A 2-relation schema is in BCNF (*show*!).
- We outline the BCNF decomposition algorithm.

BCNF decomposition Algorithm

INPUT: A relation R_0 with a set of functional dependencies S_0.
OUTPUT: A decomposition of R_0 into a collection of relations, all of which are in BCNF.
ALGORITHM is recursive and can be applied to any relation R and a set of FDs S. Initially, apply them to $R = R_0$ and $S = S_0$.

1 Check if R is in BCNF. If so return $\{R\}$ and terminate.
2 Suppose there is an FD $X \to Y$ that causes BCNF violation.
3 Compute X^+
4 Let $R_1 = X^+$ and $R_2 = X \cup (R - X^+)$
R_2 has attributes X and those attr. of R that are not in X^+.
5 Compute functional dependency projections on R_1 and R_2; let these be S_1 and S_2.
6 Recursively decompose R_1 and R_2 using this algorithm.
7 Return union of the results of these decompositions.

Good and Not so Good Decompositions

- The GOOD property: So far, we have observed that before we decompose a relation into BCNF,
  - it may exhibit anomalies.
- But after we decompose,
  - the resulting relations do not display anomalies.

Good properties of Decompositions

We would like a decomposition to have three distinct properties:

1. *Elimination of Anomalies* by decomposition process given earlier.
2. *Recoverability of Information.* Can we recover the original relation from the decomposed relations?
3. *Preservation of Dependencies*.
    1. Each decomposed relation satisfies the projection of the original set of dependencies on its schema.
    2. Would the union of the projection of the original set of dependencies on each of the decomposed schemas imply the original set of dependencies?

Dependency Preserving Decomposition

The last item is explained a bit here.

1. Let R be the relation and S be the set of dependencies.
2. Let the decomposition be $R_1, R_2, \ldots, R_p$.
3. Projected dependencies on these fragments is $S_1, S_2, \ldots, S_p$.
4. Does $S_1 \cup S_2 \cup \cdots \cup S_p$ imply S? i.e., are they equivalent?

What we know

The following facts are well-known.

1. The BCNF decomposition algorithm gives us properties (1) and (2) but not necessarily (3).
2. There is another decomposition algorithm that gives us (2) and (3) but not necessarily (1).

Lossless Join Decomposition

- We have claimed that the BCNF decomposition algorithm allows *exact recovery of information*.
- Moreover, the original relation is the natural join of its projections on the decomposed subsets of schema.
- Let us try to understand this better.
- Suppose there is a relation $R(A, B, C)$ and an FD $B \rightarrow C$.
- The FD $B \rightarrow C$ is a BCNF violation.
- The BCNF decomposition algorithm decomposes R into

$$R_1(A, B) \text{ and } R_2(B, C) \quad .$$

- Is $R_1 \bowtie R_2 = R$? Always?

Is decomposition lossless?

- $R(A, B, C)$; decomposed as $R_1(A, B)$ and $R_2(B, C)$.
- Supppose there is a tuple $t = (a, b, c)$ in R corresponding to schema (A, B, C).
  - $t[A] = a, t[B] = b$ and $t[C] = c$ for ease of notation.
- Tuple t projects as $s = (a, b)$ in $R_1(A, B)$ and as $u = (b, c)$ in $R_2(B, C)$.
- When we take the natural join of $R_1 \bowtie R_2$, these two projected tuples join.
- Because, $s[B] = b = u[B]$.
- The tuple t is therefore in $R_1 \bowtie R_2$.
- Each tuple t in R is re-constructed back by joining $R_1 \bowtie R_2$, or

$$R \subseteq \pi_{A,B}(R) \bowtie \pi_{B,C}(R) = R_1 \bowtie R_2 .$$

A lossy decomposition

R :

A	B	C
1	2	3
4	2	5

$R_1 = \pi_{A,B}(R)$:

A	B
1	2
4	2

$R_2 = \pi_{B,C}(R)$:

B	C
2	3
2	5

$R_1 \bowtie R_2$:

A	B	C
1	2	3
4	2	5
1	**2**	**5**
4	**2**	**3**

- The last two tuples are not in R. **Decomposition is lossy.**
- Why? Because attribute $B = 2$ pairs with two tuples in attribute A ($A = 1$ or $A = 4$) in R_1 and also pairs with two tuples in attribute C ($C = 3$ or $C = 5$) in R_2.
- So the join gets $2 \times 2 = 4$ tuples, last two tuples are not in R.

Lossless Decomposition

- Consider $R(A, B, C)$ and the FD $B \to C$ holds.
- The decomposition of R into $R_1(A, B)$ and $R_2(B, C)$ is **lossless** because of the FD.
- Fix an attribute value $B = b$.
- Suppose $t = (a, b, c)$ is any tuple in R with $t[B] = b$.
- The projection of t in R_1 is $s = t[R_1] = (a, b)$.
- Similarly, projection of t in R_2 is $u = t]R_2] = (b, c)$.
- Thus (a, b) joins with (b, c) to give $(a, b, c) = t$ in $R_1 \bowtie R_2$.
- **there is no other tuple in R_2 with** $B = b$. So (a, b) does not join with any other tuple u' from R_2 with $u'[B] = b$.
- This holds for each b, hence,

$$R_1 \bowtie R_2 = R \quad .$$

Condition for lossless decomposition

- *Analogously*, with $R(A, B, C)$ and FD $B \rightarrow A$, the decomposition is lossless. Why?
- Roles of A and C are just reversed.
- We have assumed A, B and C to be single attributes, but the same argument applies if they were sets of attributes X, Y, Z.
- **Sufficient Condition for Lossless Decomposition:** If $Y \rightarrow Z$ holds in R, then, the decomposition into $R_1(X, Y)$ and $R_2(Y, Z)$ is lossless, i.e.,

$$R = \pi_{X,Y}(R) \bowtie \pi_{Y,Z}(R) \ .$$

Example: Lossless or Lossy Decomposition?

Consider the following scenario.

A	B	C
a_1	b	c_1
a_2	b	c_1
a_3	b'	c_2
a_3	b'	c_3

Relation R

A	B
a_1	b
a_2	b
a_3	b'

Relation R_1

B	C
b	c_1
b'	c_2
b'	c_3

Relation R_2

- Note that neither of the dependencies $B \rightarrow A$ or $B \rightarrow C$ hold. But

$$R_1 \bowtie R_2 = R$$

Chase Test for Lossless Join

- Earlier we argued that the decomposition of $R(A, B, C)$ into $R_1(A, B)$ and $R_2(B, C)$ is a lossless decomposition under FD $B \to C$.
- We now consider a more general situation.
- Let R with schema S is decomposed into k relations $R_1, \ldots, R_k$ with schema as sets of attributes $S_1, S_2, \ldots, S_k$.
- Set of FDs that hold on R is F.
- Is it true that

$$R = \pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \pi_{S_k}(R) \; ?$$

Chase Test: Motivation

$$R = \pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \pi_{S_k}(R) \ \ ?$$

Some points:

1. Natural join is associative and commutative. Ordering of the projections in the join is unimportant.
2. Any tuple t in R is in $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \pi_{S_k}(R)$. Why?
   - The projection of $t[S_i]$ is in $\pi_{S_i}(R)$ and so t is in the result of join.
3. $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \pi_{S_k}(R) = R$ when under the set of FDs F that hold on R, every tuple in the join is in R.

The Chase test: Example

- Relation $R(A, B, C, D)$ is decomposed into subsets $S_1 = \{A, D\}, S_2 = \{A, C\}, S_3 = \{B, C, D\}$.
- Let $t = (a, b, c, d)$ be a tuple with schema (A, B, C, D). The tableau for this decomposition has 3 tuples, one per subset.

	A	B	C	D
t_1 :	a	b_1	c_1	d
t_2 :	a	b_2	c	d_2
t_3 :	a_3	b	c	d

Chase tableau

	A	B	C	D
t_1 :	a	b_1	c_1	d
t_2 :	a	b_2	c	d_2
t_3 :	a_3	b	c	d

- First row t_1 of tableau corresponds to schema $S_1 = \{A, D\}$. Further, $t[A, D] = t_1[A, D]$. Other attributes are subscripted by 1 to refer to t_1 (i.e., $t_1[B, C] = (b_1, c_1)$).
- Second row t_2 corresponds to $S_2 = \{A, C\}$. Further, $t[A, C] = t_2[A, C]$. Other attributes are subscripted as 2: $t_2[B, D] = (b_2, d_2)$.
- Third row t_3 corresponds to $S_3 = \{B, C, D\}$ and here, $t[B, C, D] = t_3[B, C, D]$. The remaining attribute $t_3[A]$ is subscripted with 3: $t_3[A] = a_3$.

Chase Algorithm: Step 1

- Suppose the given FDs are $A \to B$, $B \to C$ and $CD \to A$.
- First use the FD $A \to B$. The first two rows are equal in their A-components, hence, they must agree in the B component.
- Hence $b_1 = b_2$; after equating, denote b_2 by b_1.

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

using FD $A \to B$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

Chase Algorithm: Step 2

- Use the FD $B \to C$, the first two rows have B-attribute $= b_1$, so we set $c_1 = c$.
- We set c_1 to c. (subscripted value is set to unsubscripted value)

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

using FD $B \to C$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

Step 3

- Now use the FD $CD \to A$. Rows 1 and 3 have equal values on C, D. Hence they are equal on A.
- This sets $a_3 = a$ in tuple 3. (subscripted value replaced by unsubscripted value).

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

using $CD \to A$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

- The unsubscripted tuple $t = (a, b, c, d)$ is the last row of the tableau.
- This proves lossless decomposition. (Let's see).

Why Chase works?

- The Chase algorithm begins by projecting a tuple $t = (a, b, c, d)$ in R on each of the projection subsets, S_1, S_2, S_3.
- Each projection tuple is extended with new attribute values for all the attributes not in the subset schema.
- When the Chase algorithm results in a row that matches $t = (a, b, c, d)$, it shows that the only way to join the projected tuples from t is to reconstruct t.

Converse

- *Conversely*, suppose the Chase algorithm terminates but is unable to produce a row $t = (a, b, c, d)$.
- Consider the final tableau as an instance of $R(A, B, C, D)$.
- Clearly, $t = (a, b, c, d)$ in the join of the projections.
- But t is not in the tableau R.
- So the join

$$R \subsetneq \pi_{S_1}(R) \bowtie \cdots \bowtie \pi_{S_3}(R)$$

and hence is not lossless.

Dependency Preservation Issue: Example

- Suppose we keep a relation to keep track of which movies are currently playing in which theaters in the country.
- The relation is

  `MovieInTheater(title, theater, city).`

- We assume that the name of the theater is unique and identifies the city it is located in.

  $$\texttt{theater} \rightarrow \texttt{city}$$

- We make a (strong) assumption that in a city, there is only one theater that screens a specific movie.

  $$\texttt{title, city} \rightarrow \texttt{theater}$$

Example

- Relation: `MovieInTheater(title, theater, city)`. FDs:

  $$\texttt{theater} \rightarrow \texttt{city} \qquad \texttt{title, city} \rightarrow \texttt{theater}$$

- The design allows a theater to screen multiple movies, so $\texttt{theater} \rightarrow \texttt{title}$ is not assumed to hold.
- The keys are { `title, theater`} and { `title, city` }.
- BCNF *violation*: The FD $\texttt{theater} \rightarrow \texttt{city}$ holds but `theater` is not a superkey.
- So we decompose the relation `MovieInTheater` into two relation schemas

  `{theater, city}`
  `{title, theater}`

Problem with BCNF decomposition

- `MovieInTheater(title, theater, city)` is decomposed into relation schemas:

  `TheaterCity{theater, city}` and `PlaysIn{title, theate`

- However, the FD `title, city` $\rightarrow$ `theater` is not preserved. Why?

  `theater` $\rightarrow$ `city` is preserved in `{theater, city}`.

  No other FD is preserved.

- Decomposition is lossless join decomposition, but does not preserve `title, city` $\rightarrow$ `theater`.
- An instance example:

Non-preservation of an FD

- Suppose we use the following two relations for `PlaysIn` and `TheaterCity`.

theater	city
CinePlos	New Delhi
Akshara	New Delhi

title	theater
Baazigar	CinePlos
Baazigar	Akshara

- The only FD holds in `TheaterCity`: `theater` → `city`.
- The join of the two tables is as follows.

theater	city	title
CinePlos	New Delhi	Baazigar
Akshara	New Delhi	Baazigar

- *violates* the FD `title, city` → `theater`.

Dependencies not Preserved.

- What went wrong?
- We started with two FDs:

$$\texttt{theater} \to \texttt{city}$$
$$\texttt{title, city} \to \texttt{theater}$$

- Decomposition schema: {theater, city}, {title, theater}.
- Projected FDs:
  1. On schema {theater, city}: theater $\to$ city.
  2. On schema {title, theater}: *NONE*.
- The projected FDs do not preserve the dependency: title, city $\to$ theater.

Third Normal Form: introduction

- The solution to the example problem is to relax BCNF requirements slightly,
- for cases when a BCNF decomposition causes loss of ability to check the FDs.
- The relaxed condition is called *third normal form*.

Definition of Third Normal Form

A relation R is in *third normal form* (3NF) if:

- Whenever $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ is a non-trivial FD, either $\{A_1, A_2, \ldots, A_n\}$
  - is a superkey, *or*
  - each of the B_i's that is not among the A's *is a member of some key*.
- Note that the difference between the 3NF condition and the BCNF condition is the clause: "or, is a member of some key".
- An attribute that is a member of some key in R is said to be a *prime* attribute.
- The 3NF condition is equivalently: For each non-trivial FD, either the *LHS* is a superkey *or*, the *RHS* consists of only *prime* attributes.

Synthesis Algorithm for 3NF Schemas

We now give a method for decomposing a relation *R* into a set of relations such that

1. Each of the decomposed relations is in 3NF.
2. The decomposition has a lossless join.
3. The decomposition has the dependency-preservation property.

Synthesis Algorithm for 3NF

INPUT: A relation R and set S of functional dependencies that hold for R.

OUTPUT: 1. A decomposition of R into a collection of relations, each of which is in 3NF.
2. The decomposition satisfies both a lossless join and dependency preservation properties.

METHOD:

1. Find a minimal basis for S, say T.
2. For each FD $X \to A$ in T, use XA as the schema of one of the relations in the decomposition.
3. If none of the relation schemas from Step 2 is a superkey of R, add another relation whose schema is a key for R.

Example

Consider relation $R(A, B, C, D, E)$ with FD's

$$AB \to C, \qquad C \to B, \qquad A \to D \ .$$

1. Note that the given set of FDs is a minimal basis. (We'll check this immediately after.)
2. Since $\{A, B\}^+ = \{A, B, C, D\}$ and $\{A, C\}^+ = \{A, C, B, D\}$, both $\{A, B, E\}$ and $\{A, C, E\}$ are keys.
3. By synthesis algorithm, we get the subset schemas as follows:

$$R_1(A, B, C), \qquad R_2(B, C), \qquad R_3(A, D)$$

4. Since, none of the attributes of the schema has E, they are not keys. So we add one of the keys, say $R_4(A, B, E)$.
5. Final decomposition is

$$R_1(A, B, C), \qquad R_2(B, C), \qquad R_3(A, D), \qquad R_4(A, B, E) \ .$$

Example: Check for Minimal basis

Is this set of FDs a minimal basis?

$$AB \to C, \qquad C \to B, \qquad A \to D \ .$$

Phase 1: Check for redundant FDs.

1. Check FD $AB \to C$. Using only $C \to B$ and $A \to D$, we have, $\{A, B\}^+ = \{A, B, D\}$.
   - So $AB \to C$ is not implied, and hence it is not redundant.
2. Check FD $C \to B$. Using only $AB \to C$ and $A \to D$, we get, $\{C\}^+ = \{C\}$.
   - So, $C \to B$ is not implied and hence it is not redundant.
3. Check FD $A \to D$. Using only $AB \to C$ and $C \to A$, we get $\{A\}^+ = \{A\}$.
   - So, $A \to B$ is not implied and hence is not redundant.

Example: Check for minimal basis-II

Phase 2 check: Given FD set S, is the LHS of every FD minimal

$$AB \to C, \qquad C \to B, \qquad A \to D \ .$$

- There is only one FD $AB \to C$ that has more than one attribute on the LHS.
- We now check if either of A or B can be eliminated from the LHS while preserving equivalence with S..

1. Check if A can be removed from $AB \to C$?
   - $\{B\}^+ = \{B\}$ under S and so does not imply $B \to C$.
   - Answer is *NO*.
2. Check if B can be removed from $AB \to C$?
   - $\{A\}^+ = \{D\}$ under S and does not imply $A \to C$.
   - Answer is *NO*.

No attribute from the LHS of $AB \to C$ can be removed preserving equivalence. Hence, S is a minimal basis.

Why the 3NF Synthesis Algorithm Works

- *Lossless Join.*
  - A key K is either part of, or exactly, some decomposition schema.
  - Consider the sequence of FDs used to show that K^+ is the full schema.
  - Following this sequence of FDs, the Chase algorithm test will produce a full unsubscripted tuple.
  - Hence the join is lossless.
- *Dependency Preservation.*
  - Each FD of the minimal basis has all its attributes in some relation of the decomposition.
  - Thus, each dependency can be checked in the decomposed relations.

Synthesis Algorithm: Argument for 3NF

Third Normal Form.

- If we add a relation with a key K, this relation is surely in 3NF.
  - Reason: all its attributes are prime.
- For the relations whose schemas are the $LHS \cup RHS$ of some FD of the minimal basis,
  - The proof involves showing that a 3NF violation implies that the basis is not minimal.
  - Proof is omitted.