Design Theory for Relational Databases

csc343, Fall 2015
Fatemeh Nargesian
University of Toronto

Originally based on slides by Jeff Ullman, Diane Horton
Introduction

There are always many different schemas for a given set of data.
E.g., you could combine or divide tables.

How do you pick a schema? Which is better? What does “better” mean?

Fortunately, there are some principles to guide us.
Database Design Theory

- It allows us to improve a schema systematically.

General idea:
- Express constraints on the relationships between attributes
- Use these to decompose the relations

- Ultimately, get a schema that is in a “normal form” that guarantees good properties, such as no anomalies.

- “Normal” in the sense of conforming to a standard.

- The process of converting a schema to a normal form is called normalization.
Part I: Functional Dependency Theory
A poorly designed table

<table>
<thead>
<tr>
<th>part</th>
<th>manufacturer</th>
<th>manAddress</th>
<th>seller</th>
<th>sellerAddress</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>5.59</td>
</tr>
<tr>
<td>8624</td>
<td>Lee Valley</td>
<td>102 Vaughn</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>23.99</td>
</tr>
<tr>
<td>9141</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>12.50</td>
</tr>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>Walmart</td>
<td>5289 St Clair W</td>
<td>4.99</td>
</tr>
</tbody>
</table>

In any domain, there are relationships between attribute values.

Perhaps:

- Every part has 1 manufacturer
- Every manufacture has 1 address
- Every seller has 1 address

If so, this table will have redundant data.
Principle: Avoid redundancy

Redundant data can lead to anomalies.

<table>
<thead>
<tr>
<th>part</th>
<th>manufacturer</th>
<th>manAddress</th>
<th>seller</th>
<th>sellerAddress</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Hammers 'R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>5.59</td>
</tr>
<tr>
<td>8624</td>
<td>Lee Valley</td>
<td>102 Vaughn</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>23.99</td>
</tr>
<tr>
<td>9141</td>
<td>Hammers 'R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>12.50</td>
</tr>
<tr>
<td>1983</td>
<td>Hammers 'R Us</td>
<td>99 Pinecrest</td>
<td>Walmart</td>
<td>5289 St Clair W</td>
<td>4.99</td>
</tr>
</tbody>
</table>

- **Update anomaly**: if Hammers ‘R Us moves and we update only one tuple, the data is inconsistent.
- **Deletion anomaly**: If ABC stops selling part 8624 and Lee Valley makes only that one part, we lose track of its address.
Definition of FD

Suppose R is a relation, and X and Y are subsets of the attributes of R.

$X \rightarrow Y$ asserts that:

- If two tuples agree on all the attributes in set X, they must also agree on all the attributes in set Y.

We say that “$X \rightarrow Y$ holds in R”, or “$X$ functionally determines $Y$.”

An FD constrains what can go in a relation.
Example

\[ R(A, B, C, D) \]

The following instance of relation \( R \) violates the FD: \( BC \rightarrow D \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Can you come up with some FDs that are violated?

Find one FD that holds!
More formally...

A → B means:

∀tuples t₁, t₂,

\[(t₁[A] = t₂[A]) \Rightarrow (t₁[B] = t₂[B])\]

Or equivalently:

¬∃tuples t₁, t₂ such that

\[(t₁[A] = t₂[A]) \land (t₁[B] \neq t₂[B])\]
Generalization to multiple attributes

\[ A_1A_2 \ldots A_m \rightarrow B_1B_2 \ldots B_n \text{ means:} \]

\[ \forall \text{tuples } t_1, t_2, \]
\[ (t_1[A_1] = t_2[A_1] \land \ldots \land t_1[A_m] = t_2[A_m]) \Rightarrow \]
\[ (t_1[B_1] = t_2[B_1] \land \ldots \land t_1[B_n] = t_2[B_n]) \]

Or equivalently:

\[ \neg \exists \text{tuples } t_1, t_2 \text{ such that} \]
\[ (t_1[A_1] = t_2[A_1] \land \ldots \land t_1[A_m] = t_2[A_m]) \land \]
\[ \neg (t_1[B_1] = t_2[B_1] \land \ldots \land t_1[B_n] = t_2[B_n]) \]
Why “functional dependency”?

“dependency” because the value of \( Y \) depends on the value of \( X \).

“functional” because there is a mathematical function that takes a value for \( X \) and gives a unique value for \( Y \).

(It’s not a typical function; just a lookup.)
Coincidence or FD?

- An FD is an assertion about every instance of the relation.
- You can’t know it holds just by looking at one instance.
- You must use knowledge of the domain to determine whether an FD holds.
FDs are closely related to keys

Suppose K is a set of attributes for relation R.

Our old definition of superkey:

a set of attributes for which no two rows can have the same values.

A claim about FDs:

K is a superkey for R iff K functionally determines all of R.
FDs are a generalization of keys

Superkey: 
X \rightarrow \text{R} 

Every attribute

Functional dependency: 
X \rightarrow Y

A superkey must include all the attributes of the relation on the RHS.

An FD can have just a subset of them.
Inferring FDs

- Given a set of FDs, we can often infer further FDs.
- This will come in handy when we apply FDs to the problem of database design.
- Big task: given a set of FDs, infer every other FD that must also hold.
- Simpler task: given a set of FDs, infer whether a given FD must also hold.
Examples

If $A \rightarrow B$ and $B \rightarrow C$ hold, must $A \rightarrow C$ hold?

If $A \rightarrow H$, $C \rightarrow F$, and $F G \rightarrow A D$ hold, must $F A \rightarrow D$ hold? must $C G \rightarrow F H$ hold?

If $H \rightarrow GD$, $HD \rightarrow CE$, and $BD \rightarrow A$ hold, must $EH \rightarrow C$ hold?

Aside: we are not generating new FDs, but testing a specific possible one.
Armstrong Axioms

- Reflexivity: If Y is a subset of X, then X -> Y.
- Augmentation: If X -> Y, then XZ -> YZ.
- Transitivity: If X -> Y and Y -> Z, then X -> Z.
- Union: If X -> Y and X -> Z, then X -> YZ.
- Decomposition: If X -> YZ, then X -> Y and X -> Z.
- Pseudo transitivity: If X -> Y and YZ -> W, then XZ -> W.
Method 1: Prove an FD follows using first principles

You can prove it by referring back to:
- The FDs that you know hold, and
- The definition of functional dependency and Armstrong rules.

But the Closure Test is easier.
Method 2: Prove an FD follows using the Closure Test

Assume you know the values of the LHS attributes, and figure out everything else that is determined (X -> Y).

If it includes the RHS attributes, then you know that LHS -> RHS.

This is called the closure test.
Y is a set of attributes, S is a set of FDs.
Return the closure of Y under S.

\textbf{Attribute\_closure}(Y, S):

Initialize $Y^+$ to Y
Repeat until no more changes occur:
    If there is an FD LHS $\rightarrow$ RHS in S such that LHS is in $Y^+$:
        Add RHS to $Y^+$
Return $Y^+$
Visualizing attribute closure

Remember transitivity:
If X -> Y and Y -> Z, then X -> Z.

If LHS is in $Y^+$ and LHS -> RHS holds, we can add RHS to $Y^+$.
S is a set of FDs; LHS ->RHS is a single FD. Return true iff LHS ->RHS follows from S.

FD_follows(S, LHS ->RHS):
  \[ Y^+ = \text{Attribute\_closure}(\text{LHS}, S) \]
  return (RHS is in \( Y^+ \))
Example: Attribute Closure

\[ R(A, B, C, D, E) \]

**FDs:** \( AC \rightarrow D, B \rightarrow E, AD \rightarrow B \)

- \( AC^+ : \) ?
- \( AC \rightarrow AC: AC^+ : \{A, C\} \)
- \( AC \rightarrow D: AC^+ : \{A, C, D\} \)
- \( AC \rightarrow A, AC \rightarrow C, AC \rightarrow D, AC \rightarrow AD: \) some FDs are trivial.
- \( AD \rightarrow B: AC^+ : \{A, C, D, B\} \)
- \( D \rightarrow E: AC^+ : \{A, C, D, B, E\} \)
Example: Attribute Closure

R(A, B, C, D, E)
FDs: AC -> D, B -> E, DA -> B
Find the closure of all attributes:

\[ A^+ = \{A\} \]
\[ B^+ = \{B, E\} \]
\[ C^+ = \{C\} \]
\[ D^+ = \{D\} \]
\[ E^+ = \{E\} \]
Example: Attribute Closure - Cont’d

FDs: AC -> D, B -> E, DA -> B

Find the closure of all combinations of attributes:

- $AB^+: \{A, B, E\}$
- $AC^+: \{A, C, D, B, E\}$
- $AD^+: \{A, D, B, E\}$
- $AE^+: \{A, E\}$
- $BC^+: \{B, C, E\}$
- $BD^+: \{B, D, E\}$
- $BE^+: \{B, E\}$
- $CD^+: \{C, D\}$
- $CE^+: \{C, E\}$

- $ABC^+: \{A, B, C, D, E\}$
- $ABD^+: \{A, B, D, E\}$
- $ABE^+: \{A, B, E\}$
- $ACD^+: \{A, C, D, B, E\}$
- $ACE^+: \{A, C, E, D, B\}$
- $ADE^+: \{A, D, E, B\}$
- $...$
Example: Attribute Closure - Cont’d

FDs: AC -> D, B -> E, DA -> B

Find the closure of all combinations of attributes:

- **ABCD**: \{A, B, C, D, E\}
- **ABCE**: \{A, B, C, E, D\}
- **ACDE**: \{A, C, D, E, B\}
- **ABDE**: \{A, B, D, E\}
- **BCDE**: \{B, C, D, E\}
- **ABCDE**: \{A, B, C, D, E\}
Example: Attribute Closure and Keys

R(A, B, C, D, E)

FDs: AC -> D, B -> E, DA -> B

A⁺: {A}
B⁺: {B, E}
C⁺: {C}
D⁺: {D}
E⁺: {E}
Example: Attribute Closure and Keys

C must be a part of all keys!
Because it only appears on the LHS of FDs.

FDs: AC -> D, B -> E, DA -> B

Candidate Key

Super Key

AB⁺: {A, B, E}
AC⁺: {A, C, D, B, E}
AD⁺: {A, D, B, E}
AE⁺: {A, E}
BC⁺: {B, C, E}
BD⁺: {B, D, E}
CD⁺: {C, D}
CE⁺: {C, E}
DE⁺: {D, E}

ABC⁺: {A, B, C, D, E}
ABD⁺: {A, B, D, E}
ABE⁺: {A, B, E}
ACD⁺: {A, C, D, B, E}
ACE⁺: {A, C, E, D, B}
ADE⁺: {A, D, E, B}
Example: Attribute Closure - Cont’d

FDs: AC -> D, B -> E, DA -> B

ABCD⁺: {A, B, C, D, E}
ABCE⁺: {A, B, C, E, D}
ACDE⁺: {A, C, D, E, B}
ABDE⁺: {A, B, D, E}
BCDE⁺: {B, C, D, E}
ABCDE⁺: {A, B, C, D, E}

Super Key
Closure Test

\[ R(A, B, C, D, E) \]

\[ \text{FDs: } AC \rightarrow D, B \rightarrow E, DA \rightarrow B \]

\[ \text{Is } BC \rightarrow E \text{ followed by this set of FDs?} \]

- Yes, because we have
  - \( BC^+ : \{B, C, E\} \)

\[ \text{What does } AD^+ : \{A, D, B, E\} \text{ tell us?} \]

- \( AD \rightarrow B \)
- \( AD \rightarrow E \)

\[ \text{Find all the FDs inferred from the set of FDs.} \]
Super Keys and Candidate Keys

- **Superkey:**
  \[ X \rightarrow R \]

- where \( R \) is all attributes

- A **candidate key** is a **minimal** set of attributes necessary to identify a tuple; this is also called a minimal **superkey**.
How to find super keys and candidate keys?

Super keys
- Find the closure of all combinations (subsets) of attributes.
- The combinations whose closure is all attributes in the schema are super keys.

Candidate keys
- Minimal super keys are candidate keys.
  - Minimality: removing one attribute from the key makes it non-key.
How to verify if an FD is followed by one of more FDs.

Is $F_1$ followed by $\text{FDs} = \{F_2, F_3, \ldots\}$

Find the closure of LHS of $F_1$, using the set FDs.

If the closure contains the attributes on the RHS of $F_1$, the answer is YES.
Equivalent sets of FDs

When we write a set of FDs, we mean that all of them hold.

We can very often rewrite sets of FDs in equivalent ways.

When we say $S_1$ is equivalent to $S_2$ we mean that:

$S_1$ holds in a relation iff $S_2$ does.
How to verify if two FD sets are equivalent?

Is FD set $S_1 = \{F_1, F_2, \ldots\}$ and $S_2 = \{F'_1, F'_2, \ldots\}$ are equivalent?

- Find the closure of all combinations of attributes according to $S_1$.
- Find the closure of all combinations of attributes according to $S_2$.
- If these two closures are the same, the answer is YES.
- Or use Armstrong rules and check if you can infer $S_1$ using $S_2$ and $S_2$ using $S_1$.  

35
Exercise

Are the sets \( S_1 = \{ A \rightarrow BC \} \) and \( S_2 = \{ A \rightarrow B, A \rightarrow C \} \) equivalent?

Is \( S_2 \) inferred from \( S_1 \)?

Assume that \( A \rightarrow BC \).

\( A^+ = ABC \) – Therefore \( A \rightarrow B \) and \( A \rightarrow C \).

Is \( S_1 \) inferred from \( S_2 \)?

Assume that \( A \rightarrow B \) and \( A \rightarrow C \).

\( A^+ = ABC \) – Therefore \( A \rightarrow BC \).

\( S_1 \) and \( S_2 \) are equivalent!
Exercise

Are the sets $S_1 = \{PQ \rightarrow R\}$ and $S_2 = \{P \rightarrow Q, P \rightarrow R\}$ equivalent?

To prove that two FD sets are not equal one instance that shows one set is held the other not is enough.

This instance satisfies $PQ \rightarrow R$ but not $P \rightarrow Q, P \rightarrow R$:

S1 and S2 are not equivalent.

<table>
<thead>
<tr>
<th>Q</th>
<th>P</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Projecting FDs

Later, we will learn how to normalize a schema by decomposing relations. (This is the whole point of this theory.)

We will need to be aware of what FDs hold in the new, smaller, relations.

In other words, we must project our FDs onto the attributes of our new relations.
S is a set of FDs; L is a set of attributes.

Return the projection of S onto L:
  all FDs that follow from S and involve only attributes from L.

Project(S, L):
  Initialize T to {}.
  For each subset X of L:
    Compute $X^+$ Close X and see what we get.
  For every attribute A in $X^+$:
    If A is in L: $X \rightarrow A$ is only relevant if A is in L (we know X is).
      add $X \rightarrow A$ to T.
  Return T.
Projecting FDs - Example

Suppose we have a relation on attributes ABCDE with these FDs: \( A \rightarrow C, \ C \rightarrow E, \ E \rightarrow BD \)

Project the FDs onto attributes ABC.

\( A^+ = ACEBD: \ A \rightarrow BC \)

\( B^+ = B: \no\non\-\trivial\ FD. \)

\( C^+ = CEBD: \ C \rightarrow B \)

\( BC^+ = BCED: \This\ yields\ no\ FD\ on\ ABC. \)

We do not consider super sets of A, because we have already found out A determines all attributes.

The projection is: \( A \rightarrow BC, \ C \rightarrow B \)
Projecting FDs - Example

Suppose we have a relation on attributes ABCDE with these FDs: \(A \rightarrow C, \ C \rightarrow E, \ E \rightarrow BD\)

Project the FDs onto attributes ADE.

- \(A^+ = ACEBD: A \rightarrow DE\)
- \(D^+ = D: \) no non-trivial FD
- \(E^+ = EBD: E \rightarrow D\)
- \(DE^+ = DEB: \) This yields no FD on ADE.

We do not need to consider any supersets of A, since A determines all the attributes ADE also.

The projection is: \(A \rightarrow DE, \ E \rightarrow D\)
A few speed-ups

No need to add X -> A if A is in X itself. It’s a trivial FD.

These subsets of X won’t yield anything, so no need to compute their closures:

- the empty set
- the set of all attributes

Neither are big savings, but ...
A big speed-up

If we find $X^+ = \text{all attributes}$, we can ignore any superset of $X$.

It can only give us “weaker” FDs (with more on the LHS).

This is a big time saver!
Projection is expensive

Even with these speed-ups, projection is still expensive.

Suppose $R_1$ has $n$ attributes.

How many subsets of $R_1$ are there?
Suppose we have a relation on attributes ABCDEF with these FDs: \( ABE \rightarrow CF, \ DF \rightarrow BD, \ C \rightarrow DF, \ E \rightarrow A, \ AF \rightarrow B \)
The Splitting/Combining Rule of FDs

- Attributes on right independent of each other
  - Consider \( a, b, c \rightarrow d, e, f \)
  - “Attributes \( a, b, \) and \( c \) functionally determine \( d, e, \) and \( f \)”
  - \( \Rightarrow \) No mention of \( d \) relating to \( e \) or \( f \) directly

- Splitting rule (Useful to split up right side of FD)
  - \( abc \rightarrow def \) becomes \( abc \rightarrow d, abc \rightarrow e \) and \( abc \rightarrow f \)
  - No safe way to split left side
  - \( abc \rightarrow def \) is NOT the same as \( ab \rightarrow def \) and \( c \rightarrow def \)!

- Combining rule (Useful to combine right sides):
  - if \( abc \rightarrow d, \) \( abc \rightarrow e, \) \( abc \rightarrow f \) holds, then \( abc \rightarrow def \) holds
Splitting FDs - example

Consider the relation and FD
EmailAddress(user, domain, firstName, lastName)
user,domain -> firstName, lastName

The following hold
user,domain -> firstName
user,domain -> lastName

The following do NOT hold!
user -> firstName,lastName
domain -> firstName,lastName
Trivial FDs

Not all functional dependencies are useful

- A→A always holds
- abc→a also always holds (right side is subset of left side)

FD with an attribute on both sides is “trivial”

Simplify by removing L ∩ R from R

abc→ad becomes abc→d
Transitive Rule

The transitive rule holds for FDs

Consider the FDs: \( a \rightarrow b \) and \( b \rightarrow c \); then \( a \rightarrow c \) holds

Consider the FDs: \( ad \rightarrow b \) and \( b \rightarrow cd \); then \( ad \rightarrow cd \) holds or just \( ad \rightarrow c \)
(because of the trivial dependency rule)
Discarding Redundant FDs

- Minimal basis: opposite extreme from closure
- Given a set of FDs $F$, want to minimize $F'$ s.t.
  - $F' \subseteq F$,
  - $F'$ entails $\forall X, X \in F$

Properties of a minimal basis $F'$
- RHS is always singleton
- If any FD is removed from $F'$, $F'$ is no longer a minimal basis
- If for any FD in $F'$ we remove one or more attributes from the LHS of $F$, the result is no longer a minimal basis
Constructing a Minimal Basis

1. Split all RHS into singletons
2. \( \forall X \in F', \) test whether \( J = (F' - X)^+ \) is still equivalent to \( F^+ \)
   
   \( \Rightarrow \) Might make \( F' \) too small
3. \( \forall i \in \text{LHS}(X) \; \forall X \in F', \) let \( \text{LHS}(X') = \text{LHS}(X) - i \) test whether \( (F' - X + X')^+ \) is still equivalent to \( F^+ \)
   
   \( \Rightarrow \) Might make \( F' \) too big
4. Repeat (2) and (3) until neither makes progress
Minimal Basis: Example

- Relation R(A, B, C, D)
- FDs: F = {A→AC, B→ABC, D→ABC}
- Find the minimal basis M of F!
- Step 1: Split all RHS into singletons
- Step outcome
  - F’ = {A→A, A→C, B→A, B→B, B→C, D→A, D→B, D→C}
Minimal Basis: Example

F’ = \{A\rightarrow A, A\rightarrow C, B\rightarrow A, B\rightarrow B, B\rightarrow C, D\rightarrow A, D\rightarrow B, D\rightarrow C\}

Step 2: Remove redundant FDs.

A\rightarrow A: can be removed as trivial

A\rightarrow C: can’t be removed, as there is no other LHS with A

A+:\{A,C\}. If A\rightarrow C is removed A+:\{A\}

B\rightarrow A: can’t be removed, because F’–\{B\rightarrow A\} is B+=BC

B\rightarrow B: can be removed as trivial

B\rightarrow C: can be removed, because F’–\{B\rightarrow C\} is B+=ABC

D\rightarrow A: can be removed, because for F’–\{D\rightarrow A\} is D+=DBA

D\rightarrow B: can’t be removed, because for F’–\{D\rightarrow B\} is D+=DC

D\rightarrow C: can be removed, because for F’–\{D\rightarrow C\} is D+=DBAC

Step outcome

H = \{A\rightarrow C, B\rightarrow A, D\rightarrow B\}
Minimal Basis: Example

- Step 3: remove attributes from LHS of FDs and verify if the closure changes. If it does not change, remove the attribute.
  - H doesn’t change as all LHS in H are single attributes

- Step 4: repeat step 2 and 3.
  - H doesn’t change

Minimal Basis: M = H = {A→C, B→A, D→B}

There might be more than one minimal basis for an FD set!
Some comments on computing a minimal basis

After you identify a redundant FD, you must not use it when computing any subsequent closures (as you consider whether other FDs are redundant).
Part II:
Using FD Theory to do Database Design
Recall that poorly designed table?

<table>
<thead>
<tr>
<th>part</th>
<th>manufacturer</th>
<th>manAddress</th>
<th>seller</th>
<th>sellerAddress</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>5.59</td>
</tr>
<tr>
<td>8624</td>
<td>Lee Valley</td>
<td>102 Vaughn</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>23.99</td>
</tr>
<tr>
<td>9141</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>12.50</td>
</tr>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>Walmart</td>
<td>5289 St Clair W</td>
<td>4.99</td>
</tr>
</tbody>
</table>

- We can now express the relationships as FDs:
  - part → manufacturer
  - manufacturer → address
  - seller → address

- The FDs tell us there can be redundancy, thus the design is bad.
To improve a badly-designed schema \( R(A_1, A_2, ..., A_n) \), we will decompose it into smaller relations \( S(B_1, B_2, ..., B_m) \) and \( T(C_1, C_2, ..., C_k) \) such that:

\[
S = \pi_{B_1, B_2, ..., B_m} (R) \\
T = \pi_{C_1, C_2, ..., C_k} (R) \\
\{A_1, A_2, ..., A_n\} = \{B_1, B_2, ..., B_m\} \cup \{C_1, C_2, ..., C_k\}
\]
What we want from a decomposition?

1. No anomalies.
2. Lossless Join: It should be possible to
   a) project the original relations onto the decomposed schema
   b) then reconstruct the original by joining. We should get back exactly the original tuples.
3. Dependency Preservation: All the original FD’s should be satisfied.
What is lost in a “lossy” join?

For any decomposition, it is the case that:
\[ r \subseteq r_1 \bowtie \ldots \bowtie r_n \]

I.e., we will get back every tuple.

But it may not be the case that:
\[ r \supseteq r_1 \bowtie \ldots \bowtie r \]

I.e., we can get spurious tuples.
Splitting Relations - Example

Consider the following relation

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mary</td>
<td>mary@utsc</td>
<td>CSCD08</td>
<td>Rosenberg</td>
</tr>
</tbody>
</table>

One possible decomposition

- Students(email, name)
- Courses(course, instructor)
- Taking(studentEmail, courseName)
Lossy Join Decomposition

Consider the following relation

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mary</td>
<td>mary@utsc</td>
<td>CSCD08</td>
<td>Rosenberg</td>
</tr>
</tbody>
</table>

Students ⋈ Courses ⋈ Taking

- Mary is not taking Bretscher’s section of D08
- Xiao is not in Rosenberg’s section of D08
Information Loss with Decomposition

- Decompose R into S and T
- Consider FD a→b, with a only in S and b only in T

FD loss
- Attributes a and b no longer in same relation
- Must join T and S to enforce a→b (expensive)

Join loss
- LHS and RHS no longer in same relation, no other connection – Neither(S∩T)→S nor (S∩T)→T in F+
- Joining T and S produces bogus tuples (irreparable)

In our example:
- (\{email, course\} ∩ \{course, instructor\}) = \{course\}
- course −/−> instructor and course −/−> email
3rd Normal Form - 3NF

Consider FD X → a

Either a ∈ X OR X is a superkey OR a is prime (part of a key)

=> result, no transitive dependencies allowed

Counter example

studio → studioAddr(studioAddr depends on studio which is not a candidate key or prime)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Studio</th>
<th>Studio Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Paramount</td>
<td>Bretschier</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
</tbody>
</table>
F is a set of FDs; L is a set of attributes. Synthesize and return a schema in 3\(^{rd}\) Normal Form.

3NF\_synthesis(F, L):

1. Construct a minimal basis M for F.
2. For each FD \(X \rightarrow Y\) in M
   a. Define a new relation with schema \(X^+\).
3. Place any remaining attributes that have not been placed in any relations in step 2 in a single relation schema

If a key of R is not found in any relation, then add a trivial relation that consists of the key of R
3NF in terms of Graphs

3NF violation: transitive dependency

- **transitive FD**: dependency from `year` to `studioName`
- **lost redundant FD**: dependency from `studioName` to `studioAddr`
3NF and Lossy Decomposition

- Loss: an FD which spans two relations
- Join loss: if no transitive connection between the two nodes
- => No set of joins can reconstruct the connection
- Our 3NF example showed a lost dependency
  - title, year -> studioAddr
  - => No join loss because title->year -> studioName -> studioAddr
3NF - Example

Tournament Winners table is not in 3NF.

Note there is redundancy.

The following decomposition is in 3NF.
3NF, dependencies and join loss

Always possible to create schemas in 3NF for which lossless join, dependencies hold.

Join loss example 1:
- MovieInfo(title, year)
- StudioAddress(title, year, studioAddress)
- => Cannot enforce studioName

Join loss example 2:
- Movies(title, year, star)
- StarSalary(star, salary)
- Cannot enforce Movies \* StarSalary yields bogus tuples (irreparable)
Boyce-Codd Normal Form

We say a relation R is in BCNF if for every nontrivial FD $X \rightarrow Y$ that holds in R, X is a superkey.

Remember: nontrivial means Y is not contained in X.

Remember: a superkey doesn’t have to be minimal.
BCNF

Counterexample

CanadianAddress(street, city, province, postalCode)

Candidate keys: {street, postalCode}, {street, city, province}

FD: postalCode $\rightarrow$ city, province

Satisfies 3NF: city, province both non-prime and postalCode is prime

Violates BCNF: postalCode is not a superkey

$\Rightarrow$ Possible anomalies involving postalCode
BCNF-example

emps(emp_id, emp_name, emp_phone, dept_name, emp_city, emp_straddr)

empadds (emp_city, emp_zip, emp_straddr)

FDs:

emp_id -> (emp_name, emp_phone, dept_dname)
(emp_city, emp_straddr) -> emp_zip
emp_zip -> emp_city

The FD emp_zip -> emp_city is preserved in the relation empadds but emp_zip is not a key. The schema is not in BCNF.

The attribute emp_city is prime (there is key emp_city, emp_straddr). Hence the schema is in 3NF.
Decomposition Picture

1) Start with the LHS of the violating FD.

2) Close the LHS to get one new relation

3) Everything except the new stuff is the other new relation.
   X is in both new relations to make a connection between them.
BCNF_decomp(R, F):

If an FD $X \rightarrow Y$ in $F$ violates BCNF

Compute $X^+$. 

Replace $R$ by two relations with schemas:

$$R_1 = X^+$$

$$R_2 = R - (X^+ - X)$$

Project the FD’s $F$ onto $R_1$ and $R_2$. 

Recursively decompose $R_1$ and $R_2$ into BCNF.
Some comments on BCNF decomp

If more than one FD violates BCNF, you may decompose based on any one of them.

Because of this, there may be multiple possible results.

The new relations we create may not be in BCNF. We must recurse.

We only keep the relations at the “leaves”.

75
Speed-ups for BCNF decomposition

- Don’t need to know any keys.
- Only superkeys matter.
- And don’t need to know all superkeys.
- Only need to check whether the LHS of each FD is a superkey.
- Use the closure test (simple and fast!).
More speed-ups

- When projecting FDs onto a new relation, check each new FD:
  - Does the new relation violate BCNF because of this FD?

- If so, abort the projection.
  - You are about to discard this relation anyway (and decompose further).
Limits of decomposition

- Pick two
  - Lossless join
  - Dependency preservation
  - Anomaly free
- 3NF
  - Always allows join lossless and dependency preserving
  - May allow some anomalies
- BCNF
  - Always excludes anomalies
  - May give up one of join lossless or dependency preserving
- Use domain knowledge to choose 3NF vs. BCNF
Properties of Decompositions
What BCNF decomposition offers

1. No anomalies : ✓ (Due to no redundancy)
2. Lossless Join : ✓ (Section 3.4.1 argues this)
3. Dependency Preservation : ✗
The BCNF property does not guarantee lossless join.

- If you use the BCNF decomposition algorithm, a lossless join is guaranteed.
- If you generate a decomposition some other way, you have to check to make sure you have a lossless join.
- Even if your schema satisfies BCNF!
- We’ll learn an algorithm for this check later.
Preservation of dependencies

BCNF decomposition does not guarantee preservation of dependencies.

I.e., in the schema that results, it may be possible to create an instance that:

satisfies all the FDs in the final schema,

but violates one of the original FDs.

Why? Because the algorithm goes too far — breaks relations down too much.

[Exercise]
3NF is less strict than BCNF

3rd Normal Form (3NF) modifies the BCNF condition to be less strict.

An attribute is prime if it is a member of any key.

X -> A violates 3NF if and only if X is not a superkey, and also X is not prime.

I.e., it’s ok if X is not a superkey as long as A is prime.
F is a set of FDs; L is a set of attributes.
Synthesize and return a schema in 3\textsuperscript{rd} Normal Form.

3NF\_synthesis(F, L):

Construct a minimal basis M for F.
For each FD X -> Y in M
   Define a new relation with schema X \cup Y.
If no relation is a superkey for L
   Add a relation whose schema is some key.
3NF synthesis doesn’t “go too far”

BCNF decomposition doesn’t stop decomposing until in all relations:
- if \( X \rightarrow A \) then \( X \) is a superkey.

3NF generates relations where:
- \( X \rightarrow A \) and yet \( X \) is not a superkey, but \( A \) is at least prime.

[Example]
What a 3NF decomposition offers

1. No anomalies : ✗
2. Lossless Join : ✓
3. Dependency Preservation : ✓
   - Neither BCNF nor 3NF can guarantee all three! We must be satisfied with 2 of 3.
   - Decompose too far ⇒ can’t enforce all FDs.
   - Not far enough ⇒ can have redundancy.
   - We consider a schema “good” if it is in either BCNF or 3NF.
How can we get anomalies?

3NF synthesis guarantees that the resulting schema will be in 3rd normal form.

This allows FDs with a non-superkey on the LHS.

This allows redundancy, and thus anomalies.
How do we know...?

... that the algorithm guarantees:

- **3NF**: A property of minimal bases [see the textbook for more]
- **Preservation of dependencies**: Each FD from a minimal basis is contained in a relation, thus preserved.
- **Lossless join**: We’ll return to this once we know how to test for lossless join.
“Synthesis” vs “decomposition”

3NF synthesis:
We build up the relations in the schema from nothing.

BCNF decomposition:
We start with a bad relation schema and break it down.
If we project $R$ onto $R_1$, $R_2$, ..., $R_k$, can we recover $R$ by rejoining?

We will get all of $R$.

Any tuple in $R$ can be recovered from its projected fragments. This is guaranteed.

But will we get only $R$?

Can we get a tuple we didn’t have in $R$? This part we must check.
Aside: when we don’t need to test for lossless Join

Both BCNF decomposition and 3NF synthesis guarantee lossless join.

So we never need to test for lossless join if we have done BCNF decomposition or 3NF synthesis.

But merely satisfying BCNF or 3NF does not guarantee a lossless join!