Relational Database Design

• Finding database schemas with good properties
• Example:
  – Database for information on suppliers, parts supplied, and shipments
  – Information consists of:
    • S#: supplier number
    • SNAME: supplier name
    • SCITY: supplier city
    • P#: part number
    • PNAME: part name
    • PCITY: city where part is stored
    • QTY: quantity of shipment

The following FDs hold:
S# → SNAME SCITY
P# → PNAME PCITY
S#P# → QTY

Relational Database Design

• One possible database schema
  – BAD[S#, SNAME, SCITY, P#, PNAME, PCITY, QTY]
• Why BAD is bad:
  – redundancy
    • SCITY is determined by S#
      However, the same SCITY could appear many times with the same S# in this relation
    • Functional dependency: S# → SCITY
  – update anomalies
    • SCITY could be changed in one place but not in another, for the same S#, resulting in inconsistency
Relational Database Design

• Why BAD is bad:
  – insertion anomalies
    • cannot record supplier info unless that supplier supplies a part
  – deletion anomalies
    • by deleting parts we can lose supplier info

• Solution
  – New schema:
    S[S#, SNAME, SCITY] Key: S#
    P[P#, PNAME, PCITY] Key: P#
    SP[S#, P#, QTY] Key: S# P#
  • No other functional dependencies besides key dependencies
  • Normal forms: “nice” forms for schemas

Decomposing Relation Schemes

• Let R be a relation scheme
• A decomposition of R is a set \( \rho = \{ R_1, \ldots, R_k \} \) of relation schemas such that:
  \[ \text{att}(R) = \bigcup_{i=1}^{k} \text{att}(R_i) \]

• Example
  – \{S, P, SP\} is a decomposition of the relation schema BAD
    (see earlier example)
Conditions for a “Reasonable” Decomposition

- Do S, P, SP contain the same information as BAD?

\[
\text{BAD} = \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD})
\]

**Lossless join property**

Question: what is the connection between BAD and \( \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD}) \)?

- **A**: no connection
- **B**: \( \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD}) \subseteq \text{BAD} \)
- **C**: \( \text{BAD} \subseteq \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD}) \)
Conditions for a “Reasonable” Decomposition

- It is always true that
  \[ \text{BAD} \subseteq \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{SP}(\text{BAD}) \]
- The converse inclusion holds only because of the FDs

\[
\begin{align*}
S# &\rightarrow \text{SNAME SCITY} \\
P# &\rightarrow \text{PNAME PCITY} \\
S#P# &\rightarrow \text{QTY}
\end{align*}
\]

Conditions for a “Reasonable” Decomposition

- Can the dependencies for BAD be enforced by “local” dependencies on S, P, SP?
  - The FDs for BAD:
    \[
    \begin{align*}
    S# &\rightarrow \text{SNAME SCITY} \\
P# &\rightarrow \text{PNAME PCITY} \\
S#P# &\rightarrow \text{QTY}
    \end{align*}
    \]
  - Local FDs for S:
    \[ S# \rightarrow \text{SNAME SCITY} \]
  - Local FDs for P:
    \[ P# \rightarrow \text{PNAME PCITY} \]
  - Local FDs for SP:
    \[ S#P# \rightarrow \text{QTY} \]

So nothing is lost: the local FDs can enforce the original FDs
Not always that simple.

Example:

\begin{array}{c|ccc}
R & A & B & C \\
\hline
A \rightarrow B, A \rightarrow C, B \rightarrow C \\
\\
P & A & B \\
Q & B & C \\
A \rightarrow B & B \rightarrow C \\
\end{array}

Local FDs: A \rightarrow B, B \rightarrow C not the same as original FDs

Enough to enforce original FDs because A \rightarrow B, B \rightarrow C imply A \rightarrow C

The decomposition is dependency preserving

---

Lossless Join

- Let R be a relation schema and F a set of FDs over R. A decomposition \( \rho = \{ R_1, \ldots, R_k \} \) of R has lossless join wrt F iff, for every relation R satisfying F,

\[ R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \bowtie \ldots \bowtie \pi_{R_k}(R) \]

- Checking lossless join:
  Idea: minimize the conjunctive query

\[ \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \bowtie \ldots \bowtie \pi_{R_k}(R) \]

knowing that R satisfies the FDs F.

The result must be query returning just R itself.
Testing Lossless Join

Example: \( R = \text{BAD} \quad \rho = \{S, P, \text{SP}\} \)

SQL query for \( \pi_S(\text{BAD}) \bowtie \pi_P(\text{BAD}) \bowtie \pi_{\text{SP}}(\text{BAD}) \):

\[
\begin{align*}
\text{SELECT} & \quad s.S\#, s.SNAME, s.SCITY, \\
& \quad p.P\#, p.PNAME, p.PCITY, sp.QTY \\
\text{FROM} & \quad \text{BAD} \ s, \ \text{BAD} \ p, \ \text{BAD} \ sp \\
\text{WHERE} & \quad s.S\# = sp.S\# \text{ and } p.P\# = sp.P\#
\end{align*}
\]

Corresponding pattern:

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>SCITY</th>
<th>P#</th>
<th>PNAME</th>
<th>PCITY</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>a_2</td>
<td>a_3</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>a_4</td>
<td>a_5</td>
<td>a_6</td>
<td>--</td>
</tr>
<tr>
<td>a_1</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>a_2</td>
<td>--</td>
<td>a_7</td>
</tr>
<tr>
<td>a_1</td>
<td>a_2</td>
<td>a_3</td>
<td>a_4</td>
<td>a_5</td>
<td>a_6</td>
<td>a_7</td>
</tr>
</tbody>
</table>

\[\text{answer}\]
Testing Lossless Join

1. Compute pattern for \( Q = \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \bowtie \ldots \bowtie \pi_{R_k}(R) \)
2. Compute \( \text{CHASE}_f(Q) \)
3. Minimized \( \text{CHASE}_f(Q) \) must be \( R \)

Example: \( R = \text{BAD} \) \( \rho = \{S, P, SP\} \)

Pattern for \( \text{select} \ast \text{from BAD} \):

\[
\begin{array}{cccccccc}
S\# & S\text{NAME} & SCITY & P\# & P\text{NAME} & PCITY & QTY \\
\hline
a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
\hline
a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
\hline
\end{array}
\]

\( F = \{S\# \rightarrow \text{SCITY}, S\text{NAME}, \text{P}\# \rightarrow \text{P\text{NAME}}, \text{PCITY}, \text{P}\#S\# \rightarrow \text{QTY} \} \)

Pattern \( Q \)

\[
\begin{array}{cccccccc}
S\# & S\text{NAME} & SCITY & P\# & P\text{NAME} & PCITY & QTY \\
\hline
a_1 & a_2 & a_3 & -- & -- & -- & -- \\
\hline
-- & -- & -- & a_4 & a_5 & a_6 & -- \\
\hline
a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
\hline
\end{array}
\]

\( \text{CHASE}_f(Q) \) makes third row equal to \(<a_1, \ldots, a_7>\) so the minimized tableau is the previous pattern for \( \text{BAD} \)
Testing Lossless Join

Example: \( R = \text{BAD} \quad \rho = \{ S, P, SP \} \)

\[
F = \{ S\# \rightarrow \text{SCITY, SNAME} \\
P\# \rightarrow \text{PNAME, PCITY} \\
P\#S\# \rightarrow \text{QTY} \}
\]

Pattern \( Q \)

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>SCITY</th>
<th>P#</th>
<th>PNAME</th>
<th>PCITY</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( a_4 )</td>
<td>( a_5 )</td>
<td>( a_6 )</td>
<td>( a_7 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( a_4 )</td>
<td>( a_5 )</td>
<td>( a_6 )</td>
<td>( a_7 )</td>
</tr>
</tbody>
</table>

So \( \text{BAD} \) has lossless join with respect to \( \{ S, P, SP \} \)

Testing Lossless Join

**More concise algorithm**

Input: \( R \), decomposition \( \{ R_1, \ldots, R_k \} \), set \( F \) of FDs

1. construct a pattern using variable \( a \) for each attribute \( A \):

   for each \( R_i \) the pattern has one row with variable \( a \) for each attribute \( A \) of \( R_i \) and wildcards everywhere else

3. Chase the pattern with \( F \)

4. Output YES iff the resulting pattern has an entire row of variables
### Example

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>SCITY</th>
<th>P#</th>
<th>PNAME</th>
<th>PCITY</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a₁</td>
<td>a₂</td>
<td>a₃</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>--</td>
<td>a₄</td>
<td>a₅</td>
<td>a₆</td>
<td>--</td>
</tr>
<tr>
<td>SP</td>
<td>a₁</td>
<td>a₂</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td>a₆</td>
</tr>
</tbody>
</table>

F = \{S# \rightarrow \text{SCITY SNAME} \}

P# \rightarrow \text{PNAME PCITY}

P#S# \rightarrow \text{QTY}

Another example

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>SCITY</th>
<th>P#</th>
<th>PNAME</th>
<th>PCITY</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a₁</td>
<td>a₂</td>
<td>a₃</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>--</td>
<td>a₄</td>
<td>a₅</td>
<td>a₆</td>
<td>--</td>
</tr>
<tr>
<td>SP</td>
<td>a₁</td>
<td>a₂</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td>a₆</td>
</tr>
</tbody>
</table>
Towards dependency preservation: FD implication

- Given FDs can **imply** additional FDs
  Example:  \( A \rightarrow B \) and \( B \rightarrow C \) **imply** \( A \rightarrow C \)
  
  Every relation that satisfies \( A \rightarrow B \) and \( B \rightarrow C \) also satisfies \( A \rightarrow C \)

  Another example:
  \( A \rightarrow C, BC \rightarrow D, AD \rightarrow E \) **imply** \( AB \rightarrow E \)

---

**Definition:** A set \( F \) of FDs **implies** another FD \( X \rightarrow Y \) if every relation satisfying \( F \) also satisfies \( X \rightarrow Y \)

**Notation for “\( F \) implies \( X \rightarrow A \)”**: \( F \models X \rightarrow A \)

Everything that \( F \) implies:
\( F^+ = \{X \rightarrow Y | F \models X \rightarrow Y\} \)

**Example:** \( \{A \rightarrow B, B \rightarrow C\}^+ \) includes the following FDs:
\( A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C \)

also “trivial” FDs (that are always true):
\( A \rightarrow A, AB \rightarrow A, ABC \rightarrow A, B \rightarrow B, AB \rightarrow B, \) etc
Checking if $X \rightarrow A$ is implied by a set $F$ of FDs

Needed to compute keys, check dependency preservation, check satisfaction of normal forms, etc.

Useful to think of the closure of $X$: the set of attributes “determined” by $X$

Definition: The closure of a set of attributes $X$ with respect to a set $F$ of FDs is

$$X^+ = \{ A \mid F \models X \rightarrow A \}$$

By the definition, $X \rightarrow A \in F^+$ iff $A \in X^+$

Example of Closure Computation

- $R = ABCDEF$
- $F = \{ A \rightarrow C, BC \rightarrow D, AD \rightarrow E \}$
- $X = AB$

Computing $X^+$:

- $X^{(0)} = AB$
- $X^{(1)} = ABC$
- $X^{(2)} = ABCD$
- $X^{(3)} = ABCDE$
- $X^{(4)} = X^{(3)}$
- $X^+ = ABCDE$

To check if $X$ is key in $R$: $X^+ = \text{att}(R)$
Computing the closure of a set of attributes wrt a set of FD’s

- Let $F$ be a set of FD’s over $R$, and $X \subseteq R$. The following computes the closure $X^+$ of $X$ wrt $F$

  1. $X^{(0)} \leftarrow X$
  2. $X^{(i+1)} \leftarrow X^{(i)} \cup \{Z \mid V \rightarrow Z \in F, V \subseteq X^{(i)}\}$

  We get: $X^{(0)} \subseteq X^{(1)} \subseteq \ldots \subseteq X^{(i)} \subseteq X^{(i+1)} \subseteq \ldots \subseteq \text{att}(R)$

  Since $\text{att}(R)$ is finite, there must exist a $k \geq 0$ such that $X^{(k)} = X^{(k+1)}$

  Then, $X^+ = X^{(k)}$

Dependency Preserving Decompositions

- “Local” FDs

  if $F$ is a set of FD’s over $R$ and $X \subseteq \text{attributes}(R)$ then

  $\pi_X(F^+) = \{V \rightarrow W \in F^+ \mid V \subseteq X \text{ and } W \subseteq X\}$

  These are the FDs implied by $F$

  that apply to the set of attributes $X$ (“local” to $X$)
Dependency Preserving Decompositions

Example: \( F = \{A \rightarrow C, BC \rightarrow D, AD \rightarrow E\} \)

The FDs in \( F^+ \) that are

- local to \( AC \): \( A \rightarrow C \)
  \[ A^+ = AC \quad C^+ = C \]

- local to \( ABD \): \( AB \rightarrow D \)
  \[ A^+ = AC \quad B^+ = B \quad D^+ = D \]
  \[ AB^+ = ABCDE \quad AD^+ = ADCE \quad BD^+ = BD \]

- local to \( ABCE \): \( A \rightarrow C \quad AB \rightarrow CE \quad AE \rightarrow C \)
  \[ ABC \rightarrow E \quad ABE \rightarrow C \]
  \[ A^+ = AC \quad B^+ = B \quad C^+ = C \quad E^+ = E \]
  \[ AB^+ = ABCDE \quad AC^+ = AC \quad AE^+ = AEC \]
  \[ BC^+ = BCD \quad BE^+ = BE \quad CE^+ = CE \]
  \[ ABC^+ = ABCDE \quad ABE^+ = ABEDC \quad BCE^+ = BCED \]

Dependency Preserving Decompositions

- Definition:
  - Let \( \rho = (R_1, \ldots, R_k) \) be a decomposition for \( R \) and \( F \) a set of FD's over \( R \). Then \( \rho \) preserves \( F \) iff

  \[ \bigcup_{i=1}^{k} \pi_{R_i}(F^+) \] is equivalent to \( F \)

  In other words, all FDs in \( F \) are implied by the local FDs
Testing Preservation of Dependencies

- Naïve method:
  1. Compute $F^+$
  2. Compute $G = \bigcup_{i=1}^{k} \pi_{R_i}(F^+)$ %local fds
  3. Check that $F \subseteq G^+$

Drawback: impractical, since the size of $F^+$ can be exponential in the size of $F$.

- Improved method:
  avoid computing all of $G$
  idea: $X \rightarrow A$ is local to $R_i$ iff $AX \subseteq \text{att}(R_i)$ and $A \in X^+$

Example

$R = \{A, B, C, D\}$ \quad $\rho = \{AB, BC, CD\}$
$F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

Does $\rho$ preserve $D \rightarrow A$? (Is it implied by the local FDs?)

Decomposition:

\begin{tabular}{ccc}
  & * & * & * \\
  AB & BC & CD \\
\end{tabular}

Start with D

$D^+ = DABC$ so $D \rightarrow C$ is local to $CD$, add $C$

$C^+ = CDAB$ so $C \rightarrow B$ is local to $BC$, add $B$

$B^+ = BCDA$ so $B \rightarrow A$ is local to $AB$, add $A$ \hspace{1cm} success!

$D \rightarrow A$ is implied by local FDs
Testing Preservation of Dependencies

• Naïve method:
  1. Compute \( F^+ \)
  2. Compute \( G = \bigcup_{i=1}^k \pi_{R_i}(F^+) \) %local fds
  3. Check that \( F \subseteq G^+ \)

Drawback: impractical, since the size of \( F^+ \) can be exponential in the size of \( F \).

• Improved method:
  
  for each \( X \rightarrow Y \) in \( F \) do
  
  \[ Z := X \]
  
  while changes occur in \( Z \) do
  
  for \( i := 1 \) to \( k \) do
  
  \[ Z := Z \cup ((Z \cap R_i)^+ \cap R_i) \]

  if \( Y \nsubseteq Z \) output “no” and stop

  output “yes” (+ is wrt \( F \))

This computes the closure of \( Z \) wrt \( G \)

Full example

\( R = \{A, B, C, D\} \)
\( \rho = \{AB, BC, CD\} \)
\( F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\} \)

Does \( \rho \) preserve \( F \)?
Clearly, \( \rho \) preserves \( A \rightarrow B, B \rightarrow C, C \rightarrow D \) (they are local FDs).

Does \( \rho \) preserve \( D \rightarrow A \)?

Decomposition:

\[ \begin{array}{ccc}
AB & BC & CD \\
\end{array} \]

Start with \( D \)

\( D^+ = DABC \) so \( D \rightarrow C \) is local to \( CD \), add \( C \)

\( C^+ = CDAB \) so \( C \rightarrow B \) is local to \( BC \), add \( B \)

\( B^+ = BCDA \) so \( B \rightarrow A \) is local to \( AB \), add \( A \) \( \leftarrow \) success !

\( D \rightarrow A \) is preserved
Another example

\[ \text{Att}(R) = ABCDE \]
\[ \rho = \{\text{ABC}, \text{ADE}, \text{CE}\} \]

\[ F = \{AB \rightarrow C, C \rightarrow E, E \rightarrow C, C \rightarrow D, AB \rightarrow E\} \]

Normal Forms

• Terminology recall:

Let R be a relation schema and F a set of fd’s over attributes(R).

- **Superkey**: \( X \subseteq \text{att}(R) \) such that \( X \rightarrow \text{att}(R) \in F^+ \).
  X determines all attributes of R

- **Key**: \( X \subseteq \text{att}(R) \) such that \( X \rightarrow \text{att}(R) \in F^+ \)
  and there is no \( Y \subset X \) such that \( Y \rightarrow \text{att}(R) \in F^+ \).
  X is a minimal superkey

Example:
\[ \text{att}(R) = ABC \quad F = \{A \rightarrow B, B \rightarrow C\} \]
Superkeys: A, AB, AC, ABC
Key: A (the only one)
Purpose of normal forms

- Eliminate problems of redundancy and anomalies.
- Normal forms:
  - first, second, third, Boyce-Codd

**Boyce-Codd Normal Form (BCNF)**

A relation scheme R is in BCNF wrt a set of FD’s F over R, iff whenever \( X \rightarrow A \in F^+ \) and \( A \notin X \), \( X \) is a superkey for R.
Decomposition of a relation schema into BCNF relation schemas, with lossless join

- Any relation scheme has a decomposition into BCNF relation schemes, with lossless join
  not always dependency preserving

- Example
  \( R = \text{CITY ZIP ST} \)
  \( F = \text{CITY ST} \rightarrow \text{ZIP, ZIP} \rightarrow \text{CITY} \)
  - \( R \) has no decomposition into BCNF schemas, which preserves \( F \)
    \( (\text{CITY ST} \rightarrow \text{ZIP} \) is never preserved)  

Algorithm to obtain a lossless join decomposition of a given \( R \) wrt \( F \)

**Basic step (example):**
Suppose \( S = \text{ABC} \) only local FD: \( A \rightarrow B \)
This violates BCNF because \( A \) is not a superkey in \( S \)

Decomposition step to eliminate violation:

\[
\begin{array}{c}
\text{ABC} \\
\downarrow \\
\text{AB} \quad \text{AC}
\end{array}
\]

This has lossless join:
apply the test using the FD \( A \rightarrow B \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

85

86
Algorithm to obtain a lossless join decomposition of a given R wrt F

Start with $\rho = \{R\}$

Apply recursively the following procedure:
for each $S$ in $\rho$ not in BCNF,
let $X \rightarrow A \in F^+$ be such that $XA \subseteq S$, $A \not\in X$, $X$ is not a superkey of $S$ \[ \text{violation of BCNF} \]
replace $S$ by $S_1$ and $S_2$, where
$S_1 = XA$
$S_2 = X(S - A)$
until all relation schemes are in BCNF

Note: \[
\begin{array}{c}
S \\
S_1 \\
S_2
\end{array}
\]
this decomposition has lossless join:
$X \rightarrow A \in F^+$
$X$ is a key for $S_1$

Example

$R = C \ T \ H \ R \ S \ G$
$F: C \rightarrow T$
$H \rightarrow C$
$H \rightarrow R$
$C \rightarrow G$
$H \rightarrow R$

What are the keys?
only key: HS
Example

FDs: $C \rightarrow T$, $CS \rightarrow G$, $HR \rightarrow C$, $HS \rightarrow R$, $TH \rightarrow R$

**CTHRSG**
- violation: $CS \rightarrow G$  $(CS)^+ = CSTG$

**CSG**
- no violation: $C+ = CT$, $S+ = S$, $G+ = G$

**CT**
- violation: $C \rightarrow T$  $C+ = CT$

**CHRS**
- no violation: 2 attributes

**CHR**
- violation: $CH \rightarrow R$  $(CH)^+ = CHTR$

**CHS**
- no violation: $C+ = CT$, $H+ = H$, $R+ = R$, $S+ = S$

**Resulting decomposition:** CSG, CT, CHR, CHS

Remarks (drawbacks)

- Decomposition is not unique
  - CHRS could be decomposed into CHR and CHS or CHR and RHS
- The decomposition does not preserve $TH \rightarrow R$

run dependency preservation algorithm

$F = C \rightarrow T$  $CS \rightarrow G$, $HR \rightarrow C$, $HS \rightarrow R$, $TH \rightarrow R$

for the previous BCNF decomposition:

```sql
CSG  CT  CHR  CHS
```

the local closure of TH is TH
so $TH \rightarrow R$ is not preserved
• Is there an efficient algorithm for BCNF decomposition?
  – Most likely NO:

    It is NP-complete to decide whether a relation R is in BCNF with respect to a set of fd’s.

    \[ \Downarrow \]

    Any algorithm for BCNF is most likely exponential

• Problem with BCNF:
  – Not every relation schema can be decomposed into BCNF relation schemas which have lossless join and preserve the dependencies.

• Third Normal Form (3NF)
  – A relation scheme R is in Third Normal Form wrt a set F of fd’s over R, if whenever \( X \rightarrow A \) holds in R and \( A \notin X \) then either X is a superkey or A is prime

\[ A \in att(R) \text{ is prime: } A \in K \text{ for some key } K \]

3NF is weaker than BCNF
Decomposition of a relation schema into BCNF relation schemas, with lossless join

- Example
  \[ R = \text{CITY ZIP ST} \]
  \[ F = \text{CITY ST} \rightarrow \text{ZIP}, \text{ZIP} \rightarrow \text{CITY} \]

R is in 3NF but not in BCNF

Violation of BCNF: ZIP \rightarrow \text{CITY}
However, CITY belongs to the key CITY ST so this satisfies 3NF

Computing a 3NF decomposition

Three main steps:
1. Simplify the set of FDs (eliminate redundancies)
2. Construct a first-cut decomposition from remaining FDs
e.g.: from A \rightarrow B make a relation AB
      dependency preserving, not always lossless join
3. If needed, modify decomposition to ensure lossless join
Eliminating redundancies in the FDs

1. Rewrite the FDs with single attributes on RHS
   ex: replace \( AB \rightarrow CD \) with \( AB \rightarrow C \) and \( AB \rightarrow D \)

2. Eliminate redundant FDs
   ex: \( F = \{ A \rightarrow C, \ A \rightarrow B, \ B \rightarrow C \} \)
   \( A \rightarrow C \) is redundant (it is implied by \( A \rightarrow B \) and \( B \rightarrow C \))

3. Eliminate redundant attributes from LHS of FDs
   ex: \( F = \{ A \rightarrow B, \ AB \rightarrow C \} \)
   \( B \) is redundant in \( AB \rightarrow C \) because \( A \) by itself determines \( C \) (\( A \rightarrow C \) is implied by \( F \))

Example minimization

Attributes: ABCDEI
FDs: \( A \rightarrow C, AB \rightarrow C, C \rightarrow DI, CD \rightarrow I, EC \rightarrow AB, EI \rightarrow C \)
First-cut 3NF decomposition

\( R \) a schema

\( F \) minimal set of FD’s over \( R \) (no redundancies)

\( \rho = \{XA_1 \ldots A_m \mid X \rightarrow A_i \in F\} \cup \{B \in R \mid B \text{ does not occur in } F\} \)

- Example
  - \( R = \text{CTHRSG} \) (see earlier example)
  - \( F = C \rightarrow T \ CS \rightarrow G \)
    \( \begin{align*}
    HR \rightarrow C & \quad HS \rightarrow R & \text{(minimal)} \\
    HT \rightarrow R &
    \end{align*} \)
  - Then \( \rho = \{CT, HRC, HTR, CSG, HRS\} \)

First-cut 3NF decomposition

- **Theorem.** \( \rho \) preserves \( F \) and each \( R_i \) in \( \rho \) is in 3NF wrt \( \pi_{R_i}(F^+) \).

Why is each relation in 3NF?

Proof idea: Suppose \( XA \) is constructed from \( X \rightarrow A \) in \( F \)

Let \( Y \rightarrow B \) be local to \( XA \). Two cases:

1. \( B = A \). Then \( Y = X \) and \( Y \) is a superkey (if \( Y \subset X \) then attributes in \( X - Y \) are redundant)
2. \( B \neq A \), so \( B \in X \) and \( B \) is prime (\( X \) must be a key, otherwise \( X \rightarrow A \) has redundant attributes on LHS).
Second cut: ensure lossless join

• To obtain decomposition that also has lossless join:
  – add to $\rho$ a set $K$ where $K$ is a key for $R$
    …unless there already is a “piece” in the decomposition
    whose attributes form a superkey for $R$

• Theorem $\rho \cup \{K\}$ is a 3NF decomposition which is
  dependency preserving and has lossless join.
  Proof idea: chasing the pattern for $\rho \cup \{K\}$ produces
  answer variables in the entire row for $K$. Apply the FDs
  in the same order used in the computation of $K^+$

Example

• $R: ABCDE$  $F = \{A \rightarrow C, BC \rightarrow D, AD \rightarrow E\}$
  $\rho = \{AC, BCD, ADE\}$

• None of AC, BCD, ADE is a superkey:
  $AC^+ = AC$, $BCD^+ = BCD$, $ADE^+ = ADEC$

• AB is a key, add it to $\rho$

• Claim: $\rho \cup \{AB\}$ has lossless join (see next slide)
Test for lossless join

\[ F = \{A \rightarrow C, BC \rightarrow D, AD \rightarrow E\} \]

Pattern for \{\{AC, BCD, ADE, AB\}:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
<td>c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>-</td>
</tr>
<tr>
<td>a</td>
<td>-</td>
<td>-</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

FDs used in computation of \(AB^+\) (in order):

- \(A \rightarrow C\), \(BC \rightarrow D\), \(AD \rightarrow E\)
- \(AB, ABC, ABCD, ABCDE\)

A (simplified) overview of schema design using FDs

1. Choose attributes in \(R\)

2. Specify dependencies \(F\)
   
   (tool: “Armstrong relations” – relations satisfying exactly \(F^+\))

3. Find a lossless-join, dependency preserving decomposition into Normal Form schemas
   - BCNF, if possible
   - 3NF, if not BCNF
Attributes vs. data

Example: keep information about employee names and the departments in which they work
Departments: PCs, Toys, Candy
Two options:

1  Emp name   dept
2  Emp name   PCs   Toys   Candy

Some criteria:
• Are departments stable?
• Sparse vs. dense tables
• Ease of querying

Find the departments in which Joe works

A: easier with 1
B: easier with 2

Find the employees who work in every department

A: easier with 1
B: easier with 2
Armstrong relation example

<table>
<thead>
<tr>
<th>Schedule</th>
<th>theater</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paloma</td>
<td>Casablanca</td>
<td></td>
</tr>
<tr>
<td>Hillcrest</td>
<td>Casablanca</td>
<td></td>
</tr>
</tbody>
</table>

F = \{theater \rightarrow title\}

Armstrong relation for F: satisfies exactly the FDs in F⁺

- satisfies theater \rightarrow title
- violates title \rightarrow theater

Full normalization example

Attributes: broker, office, investor, stock, quantity, dividend

<table>
<thead>
<tr>
<th>B</th>
<th>O</th>
<th>I</th>
<th>S</th>
<th>Q</th>
<th>D</th>
</tr>
</thead>
</table>

FDs: S \rightarrow D, I \rightarrow B, IS \rightarrow Q, B \rightarrow O
<table>
<thead>
<tr>
<th>Attributes: broker, office, investor, stock, quantity, dividend</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B \quad O \quad I \quad S \quad Q \quad D )</td>
</tr>
<tr>
<td>\text{FDs:} \ S \rightarrow D, I \rightarrow B, IS \rightarrow Q, B \rightarrow O</td>
</tr>
</tbody>
</table>

107

<table>
<thead>
<tr>
<th>Attributes: broker, office, investor, stock, quantity, dividend</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B \quad O \quad I \quad S \quad Q \quad D )</td>
</tr>
<tr>
<td>\text{FDs:} \ S \rightarrow D, I \rightarrow B, IS \rightarrow Q, B \rightarrow O</td>
</tr>
</tbody>
</table>

108
A glimpse beyond FDs

• Example

| Movie | title | director | actor |

  – Suppose each movie may have several directors and several actors
  – then there are no fds (so this is in BCNF)
  – redundancy still exists

Directors and actors are independent info for same title

Better design:  Directors | title | director | Actors | title | actor

Captured by “multi-valued dependencies”

Directors and actors are independent info for same title

Taking in to account MVDs results in an extension of BCNF called 4th Normal Form
Alternative approach

- Start with an Entity-Relationship (ER) diagram
- Translate into corresponding relational schema
- Identify functional dependencies
- Use design theory to further improve and bring to BCNF or 3NF or 4NF

Example: piece of an E-R diagram
Corresponding relational schema

| customer   | customer-id | customer-name | customer-zip | customer-city |
| account    | account-number | balance |
| depositor  | customer-id | account-number | access date |
First-cut relational schema

- branch = (branch_name, branch_city, assets)
- customer = (customer_id, customer_name, customer_street, customer_city)
- loan = (loan_number, amount)
- account = (account_number, balance)
- employee = (employee_id, employee_name, telephone_number, start_date)
- dependent_name = (employee_id, dname)
- account_branch = (account_number, branch_name)
- loan_branch = (loan_number, branch_name)
- borrower = (customer_id, loan_number)
- depositor = (customer_id, account_number)
- cust_banker = (customer_id, employee_id, type)
- works_for = (worker_employee_id, manager_employee_id)
- payment = (loan_number, payment_number, payment_date, payment_amount)
- savings_account = (account_number, interest_rate)
- checking_account = (account_number, overdraft_amount)