Databases - Normalization III
This lecture describes 3rd normal form.
BCNF - recap

The BCNF decomposition of a relation is derived by a recursive algorithm. A *lossless-join* decomposition is derived which may not be *dependency preserving*.

The decomposition is too restrictive. To make the decomposition in the previous example dependency preserving we can cover the FD $JP \rightarrow C$ by adding its attributes as a relation

\[ R_1 = CSJDQV \quad R_2 = SDP \quad R_3 = JPC \]

We have added the required FD involving key attributes that were prohibited by BCNF.
A relational schema is in $3NF$ if for every functional dependency $X \rightarrow A$ (where $X$ is a subset of the attributes and $A$ is a single attribute) either

1. $A \in X$ (that is, $X \rightarrow A$ is a trivial FD), or
2. $X$ is a superkey, or
3. $A$ is part of some key for $R$.

Thus the definition of $3NF$ permits a few additional FDs involving key attributes that are prohibited by BCNF.
The contracts example

The contracts example was not in BCNF because of the FD

\[ SD \rightarrow P \]

However as \( P \) is part of a key \((JP)\), this FD does not violate the conditions for 3NF. Therefore neither of the two given FDs violate the conditions for 3NF.

Checking all the other (implied) FDs shows that contracts is already in 3NF and thus need not be decomposed further.
Properties of 3NF

The most important property of 3NF is the result that there is always a lossless-join and dependency-preserving decomposition of any relational schema into 3NF.

While not conceptually difficult, the algorithm to decompose a relation schema into 3NF is quite fiddly and much more complicated than the simple algorithm for decomposition into BCNF.
3NF Synthesis

The BCNF algorithm is from the perspective of deriving a *lossless-join* decomposition, and *dependency preservation* was addressed by adding extra relation schemas.

An alternative approach involves *synthesis* that takes the attributes of the original relation $R$ and the *minimal cover* $F$ of the FDs and adds a relation schema $XA$ for each FD $X \rightarrow A$ in $F$.

The resulting collection of relations is 3NF and preserves all FDs. *If* it is not *lossless-join*, it is made so by adding a relation that contains those attributes that appear in some key.

(N Spadaccini 2010 and W Liu 2012)
The two approaches

The decomposition algorithm derives the relations while maintaining the keys, thus emphasizing the lossless-join property, and deals with dependency preservation by adding the necessary relations.

The synthesis algorithm build the relations from the set of FDs, emphasizing the dependencies, and then deals with lossless-join issues after the fact by adding the necessary relation.

The latter is an algorithm that has \textit{polynomial complexity}. 
The Synthesis Algorithm

Given a relation $R$ and a set of functional dependencies $X \rightarrow Y$ where $X$ and $Y$ are subsets of the attributes of $R$.

1. Compute the closures of the FDs.
2. Use 1 to find all the candidate keys of the relation $R$.
3. Find the minimal cover of the relation $R$.
4. Use 3 to produce $R_i$ decompositions in 3NF.
5. If no $R_i$ is a super key of $R$, add a relation containing the key.

(N Spadaccini 2010 and W Liu 2012)
FD closures

1. Compute the closures of the FDs.

For each $X \rightarrow Y$, derive $(X)^+$ the set of attributes of $R$ covered by $X \rightarrow Y$. 
Candidate keys

2. Use ① to find all the candidate keys of the relation $R$. If any $(X)^+$ covers all the attributes of $R$ then $X$ is a key. It is possibly a super key with redundant attributes contained within $X$ (these will be obvious).

Otherwise find the $(X)^+$ that maximally covers $R$ and add to $X$ attributes of $R$ missing from the closure.
Minimal cover

Find the *minimal* cover of the relation $R$.

**Algorithm:**

- **Step 1** Decompose all the FDs to the form $X \rightarrow A$, where $A$ is a single attribute.
- **Step 2** Eliminate redundant attributes from the LHS.
  - If $XB \rightarrow A$, where $B$ is a single attribute and $X \rightarrow A$ is entailed within the set, then $B$ was unnecessary.
  - If LHS has more than one attribute, compute closure of each set that leaves out one attribute. If it includes the right side, remove the extra attribute in the functional dependency.
- **Step 3** Delete the redundant FDs.
  - For each dependency, compute the closure of the determinant attributes against all the other dependencies except the one you’re testing. If you find the attribute on the right side in the closure, you can leave that dependency out.
3NF relations

Use \( \mathcal{R} \) to produce \( R_i \) decompositions in 3NF.

Algorithm:

- **Step 1** Combine all FDs with the same LHS:
  
  For each \( X \) such that \( X \rightarrow A, X \rightarrow B \ldots \)
  
  \( \Rightarrow X \rightarrow AB \ldots \).

- **Step 2** Form a relation \( R_i \) of all attributes in each FD:
  
  For each \( X \rightarrow ABC \), form
  
  \( \Rightarrow \{X, A, B, C\} \).
3NF relations

If no $R_i$ is a super key of $R$, add a relation containing the key.

If $UVW$ is a key for $R$, and no relation $R_i$ formed in the 3NF decomposition contains a super key of $R$, then add $\{U, V, W\}$ as a relation.
Exercise

Consider \( R = \{A, B, C, D, E, G\} \) and the following FDs

1. \( AB \rightarrow C \)
2. \( C \rightarrow A \)
3. \( BC \rightarrow D \)
4. \( ACD \rightarrow B \)
5. \( D \rightarrow EG \)
6. \( BE \rightarrow C \)
7. \( CG \rightarrow BD \)
8. \( CE \rightarrow AG \)

\( (AB)^+ = ABCDEG \)
\( (C)^+ = AC \)
\( (BC)^+ = ABCDEG \)
\( (ACD)^+ = ABCDEG \)
\( (D)^+ = DEG \)
\( (BE)^+ = ABCDEG \)
\( (CG)^+ = ABCDEG \)
\( (CE)^+ = ABCDEG \)

\( AB \) is a key
\( BC \) is a key
\( ACD \) is a super key
\( BE \) is a key
\( CG \) is a key
\( CE \) is a key

Applying Armstrong’s Axioms gives \( BD \) and \( CD \) are also keys.
Generate minimal cover - Working

Original FDs

1. $AB \rightarrow C$
2. $C \rightarrow A$
3. $BC \rightarrow D$
4. $ACD \rightarrow B$
5. $D \rightarrow EG$
6. $BE \rightarrow C$
7. $CG \rightarrow BD$
8. $CE \rightarrow AG$

single RHS

1. $AB \rightarrow C$
2. $C \rightarrow A$
3. $BC \rightarrow D$
4. $ACD \rightarrow B$
5. $D \rightarrow E$
6. $D \rightarrow G$
7. $BE \rightarrow C$
8. $CG \rightarrow B$
9. $CG \rightarrow D$
10. $CE \rightarrow A$
11. $CE \rightarrow G$

redundant LHS

1. $AB \rightarrow C$
2. $C \rightarrow A$
3. $BC \rightarrow D$
4. $ACD \rightarrow B$
5. $D \rightarrow E$
6. $D \rightarrow G$
7. $BE \rightarrow C$
8. $CG \rightarrow B$
9. $CG \rightarrow D$
10. $CE \rightarrow A$
11. $CE \rightarrow G$

Find $AB^+$ wrt the rest of FDs

minimal FDs

1. $AB^+ = \{A, B^3\}$
   - $C \notin AB^+$ keep 1
2. $C \rightarrow A$
3. $BC \rightarrow D$
4. $CD^+ = \{C, D, A, E, G, B\}$
   - $B \in CD^+$, remove 4
5. $D \rightarrow E$
6. $D \rightarrow G$
7. $BE \rightarrow C$
8. $CG \rightarrow B$
9. $CG \rightarrow D$
10. $CE \rightarrow A$
11. $CE \rightarrow G$

(N Spadaccini 2010 and W Liu 2012)
Generate minimal cover

<table>
<thead>
<tr>
<th>single RHS</th>
<th>redundant LHS</th>
<th>minimal FDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $AB \rightarrow C$</td>
<td>1. $AB \rightarrow C$</td>
<td>1. $AB \rightarrow C$</td>
</tr>
<tr>
<td>2. $C \rightarrow A$</td>
<td>2. $C \rightarrow A$</td>
<td>2. $C \rightarrow A$</td>
</tr>
<tr>
<td>3. $BC \rightarrow D$</td>
<td>3. $BC \rightarrow D$</td>
<td>3. $BC \rightarrow D$</td>
</tr>
<tr>
<td>4. $ACD \rightarrow B$</td>
<td>4. $ACD \rightarrow B$</td>
<td>4. $CD \rightarrow B$</td>
</tr>
<tr>
<td>5. $D \rightarrow E$</td>
<td>5. $D \rightarrow E$</td>
<td>5. $D \rightarrow E$</td>
</tr>
<tr>
<td>6. $D \rightarrow G$</td>
<td>6. $D \rightarrow G$</td>
<td>6. $D \rightarrow G$</td>
</tr>
<tr>
<td>7. $BE \rightarrow C$</td>
<td>7. $BE \rightarrow C$</td>
<td>7. $BE \rightarrow C$</td>
</tr>
<tr>
<td>8. $CG \rightarrow B$</td>
<td>8. $CG \rightarrow B$</td>
<td>8. $CG \rightarrow B$</td>
</tr>
<tr>
<td>9. $CG \rightarrow D$</td>
<td>9. $CG \rightarrow D$</td>
<td>9. $CG \rightarrow D$</td>
</tr>
<tr>
<td>10. $CE \rightarrow A$</td>
<td>10. $CE \rightarrow A$</td>
<td>10. $CE \rightarrow A$</td>
</tr>
<tr>
<td>11. $CE \rightarrow G$</td>
<td>11. $CE \rightarrow G$</td>
<td>11. $CE \rightarrow G$</td>
</tr>
</tbody>
</table>

Duplications:
- delete 8
- delete 10 = 2

(N Spadaccini 2010 and W Liu 2012)
The synthesis process

Combine same LHS

1. \( AB \rightarrow C \)
2. \( C \rightarrow A \)
3. \( BC \rightarrow D \)
4. \( CD \rightarrow B \)
5. \( D \rightarrow E, D \rightarrow G \)
6. \( BE \rightarrow C \)
7. \( CG \rightarrow D \)
8. \( CE \rightarrow G \)

Form relations

1. \( \{A, B, C\} \)
2. \( \{C, A\} \)
3. \( \{B, C, D\} \)
4. \( \{C, D, B\} \)
5. \( \{D, E, G\} \)
6. \( \{B, E, C\} \)
7. \( \{C, G, D\} \)
8. \( \{C, E, G\} \)

The 3NF decomposition

1. \( \{A, B, C\} \) is a proper subset of 2
2. \( \{C, A\} \) is a subset of 3
3. \( \{B, C, D\} \)
4. \( \{C, D, B\} \) is a subset of 5
5. \( \{D, E, G\} \)
6. \( \{B, E, C\} \)
7. \( \{C, G, D\} \)
8. \( \{C, E, G\} \)
Final 3NF decomposition

\{A, B, C\}\{B, C, D\}\{D, E, G\}\{B, E, C\}\{C, G, D\}\{C, E, G\}.

Super key is contained in at least one relation.

We have looked at how to confirm the decomposition is a lossless-join when an original relation \( R \) is decomposed into two \( R_1 \) and \( R_2 \).

- Simply check the shared attribute(s) of \( R_1 \) and \( R_2 \) to see if it is a superkey for either \( R_1 \) or \( R_2 \).

However when decomposed into more than two relations, we will need to use the algorithm described in the next three slides to check for lossless decomposition.
Definition: A decomposition \( D = \{R_1, R_2, ..., R_m\} \) of \( R \) has the lossless (nonadditive) join property with respect to the set of dependencies \( F \) on \( R \) if, for every relation state \( R_i \) of \( R \) that satisfies \( F \), the natural join of all the relations in \( D \):

\[
R_1 \bowtie R_2 \ldots \bowtie R_m = R
\]

Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for “loss of information” a better term is “addition of spurious information”.

(N Spadaccini 2010 and W Liu 2012)
Algorithm for Testing Lossless Join Property

- **Step 1** Create an initial matrix with one row for each decomposed relation \( R_i \), and one column for each attribute \( A_j \) in the original relation \( R \).

- **Step 2** For the \( i \)th row, if the \( j \)th attribute is in \( R_i \), cell \( (i,j) \) should contain value \( a_j \), otherwise, \( b_{ij} \).

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
& A & B & C & D & E & G \\
\hline
R_1\{A, B, C\} & a_1 & a_2 & a_3 & b_{14} & b_{15} & b_{16} \\
R_2\{B, C, D\} & b_{21} & a_2 & a_3 & b_{24} & b_{25} & b_{26} \\
R_3\{D, E, G\} & b_{31} & b_{32} & b_{33} & a_4 & a_5 & a_6 \\
R_4\{B, E, C\} & b_{41} & a_2 & a_3 & b_{44} & a_5 & b_{46} \\
R_5\{C, G, D\} & b_{51} & b_{52} & a_3 & a_4 & b_{55} & a_6 \\
R_6\{C, E, G\} & b_{61} & b_{62} & a_3 & b_{64} & a_5 & a_6 \\
\hline
\end{array}
\]
Algorithm for Testing Lossless Join Property (contd.)

- **Step 3** Apply functional dependencies to replace the $b_{ij}$s with $a_{js}$.
- **Step 4** If there is a row contains all $a_{js}$, the decomposition is lossless.

\[
R: \begin{array}{ccccccc}
A & B & C & D & E & G \\
\hline
a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\
\end{array}
\]

Successfully reconstructed

1. $AB \rightarrow C$
2. $C \rightarrow A$
3. $BC \rightarrow D$
4. $CD \rightarrow B$
5. $D \rightarrow E$
6. $D \rightarrow G$
7. $BE \rightarrow C$
8. $CG \rightarrow D$
9. $CE \rightarrow G$

(N Spadaccini 2010 and W Liu 2012)
Manual normalization

Data typically employed in a business is usually presented as below.

<table>
<thead>
<tr>
<th>Order_ID</th>
<th>Order Date</th>
<th>CustomerID</th>
<th>Customer Name</th>
<th>Customer Address</th>
<th>Product ID</th>
<th>Product Description</th>
<th>Material</th>
<th>Unit Price</th>
<th>Ordered Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>04/05/2010</td>
<td>2</td>
<td>FurnCo</td>
<td>35 S St, NY</td>
<td>21</td>
<td>Sofa</td>
<td>Leather</td>
<td>2500.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>TV Bench</td>
<td>Pine</td>
<td>300.00</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td>Chair</td>
<td>Mahogany</td>
<td>500.00</td>
<td>6</td>
</tr>
<tr>
<td>6789</td>
<td>05/05/2010</td>
<td>9</td>
<td>CheapAs</td>
<td>35 N St, LA</td>
<td>17</td>
<td>Chair</td>
<td>Mahogany</td>
<td>500.00</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>Desk</td>
<td>Particle</td>
<td>650.00</td>
<td>1</td>
</tr>
</tbody>
</table>

However to be useful in a database the data needs to be normalized.
First normal form – 1NF

First normal form (1NF) requires attribute values be *atomic* — that is, individual values rather than lists or sets and that a primary key is defined.

This table is in 1NF. The primary key is (Order ID, Product ID).

<table>
<thead>
<tr>
<th>Order_ID</th>
<th>Order Date</th>
<th>CustomerID</th>
<th>Customer Name</th>
<th>Customer Address</th>
<th>Product ID</th>
<th>Product Description</th>
<th>Material</th>
<th>Unit Price</th>
<th>Ordered Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>04/05/2010</td>
<td>2</td>
<td>FurnCo</td>
<td>35 S St, NY</td>
<td>21</td>
<td>Sofa</td>
<td>Leather</td>
<td>2500.00</td>
<td>1</td>
</tr>
<tr>
<td>1234</td>
<td>04/05/2010</td>
<td>2</td>
<td>FurnCo</td>
<td>35 S St, NY</td>
<td>9</td>
<td>TV Bench</td>
<td>Pine</td>
<td>300.00</td>
<td>4</td>
</tr>
<tr>
<td>1234</td>
<td>04/05/2010</td>
<td>2</td>
<td>FurnCo</td>
<td>35 S St, NY</td>
<td>17</td>
<td>Chair</td>
<td>Mahogany</td>
<td>500.00</td>
<td>6</td>
</tr>
<tr>
<td>6789</td>
<td>05/05/2010</td>
<td>9</td>
<td>CheapAs</td>
<td>35 N St, LA</td>
<td>17</td>
<td>Chair</td>
<td>Mahogany</td>
<td>500.00</td>
<td>6</td>
</tr>
<tr>
<td>6789</td>
<td>05/05/2010</td>
<td>9</td>
<td>CheapAs</td>
<td>35 N St, LA</td>
<td>10</td>
<td>Desk</td>
<td>Particle</td>
<td>650.00</td>
<td>1</td>
</tr>
</tbody>
</table>
A relation in first normal form (1NF) has the above functional dependencies.

A relation is in Second normal form (2NF) if it is in 1NF and contains \textbf{no} partial dependencies.

A \textit{partial functional dependency} exists when a \textit{nonkey} attribute is functionally dependent on part of, but not all of, the primary key.
Second normal form – 2NF

2NF is achieved by creating three (3) relations with the following attributes and keys.

- **Order**
  - `Order_ID`
  - `Order Date`
  - `CustomerID`
  - `Customer Name`
  - `Customer Address`

- **Order Details**
  - `Order_ID`
  - `Product ID`
  - `Ordered Quantity`

- **Product**
  - `Product ID`
  - `Product Description`
  - `Material`
  - `Unit Price`
Third normal form – 3NF

A relation in second normal form (2NF) still has the above functional dependencies.

A relation is in Third normal form (3NF) if it is in 2NF and contains no transitive dependencies.

A *transitive functional dependency* in a relation is a functional dependency between two or more *non-key* attributes.
Third normal form – 3NF

3NF is achieved by creating an additional relation with the following attributes and keys.

![Diagram showing attributes and keys in 3NF](image)
Practical Use of Normalization

In practice, the DBA should be *guided* by the theory of normalization, but not slavishly follow it.

When the ER diagram is converted into tables, any functional dependencies should be identified and a decision made as to whether the tables should be normalized or not.

Occasionally it will even be appropriate to *de-normalize* data if the most frequently occurring queries involve expensive joins of the normalized tables.
Multi-valued Attributes

A *Single-valued* attribute of an entity instance has just one value. This has been the typical scenario we have discussed so far and know how this is handled in a schema definition.

A *Multi-valued* attribute is an attribute that can take on more than one value for a given entity instance. Diagrammatically this is indicated in an ERD by a double edged ellipse.

![ERD Diagram](image-url)
Multi-valued Attributes

The mapping of an entity with a *multi-valued* attribute is achieved by the following.

The first relation contains all the attributes of the entity type except the *multi-valued* attribute. The second relation contains the attributes that form the primary key of the second relation. The first of these attributes is the primary key of the first relation, and hence it is a foreign key.

(N Spadaccini 2010 and W Liu 2012)