Databases -Normalization III

This lecture

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

$$R_1 = CSJDQV \qquad R_2 = SDP \qquad R_3 = JPC$$

We have added the required FD involving key attributes that were prohibited by BCNF.

3NF

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

- $A \in X$ (that is, $X \rightarrow A$ is a trivial FD), or
- X is a superkey, or
- 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 $X \to A$ 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.

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 *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.

FD closures

1. Compute the closures of the FDs.

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

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

3. 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

4. Use ③ to produce R_i decompositions in 3NF.

ALGORITHM:

- **Step 1** Combine all FDs with the same *LHS*

  For each X such that $X \to A$, $X \to B$ ...
  $\Rightarrow X \to AB\ldots$.

- **Step 2** Form a relation R_i of all attributes in each FD.

  For each $X \to ABC$, *form*
  $\Rightarrow \{X, A, B, C\}$.

3NF relations

5. 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. $AB \rightarrow C$	• $(AB)^+ = ABCDEG$	• AB is a key
2. $C \rightarrow A$	• $(C)^+ = AC$	•
3. $BC \rightarrow D$	• $(BC)^+ = ABCDEG$	• BC is a key
4. $ACD \rightarrow B$	• $(ACD)^+ = ABCDEG$	• ACD is a super key
5. $D \rightarrow EG$	• $(D)^+ = DEG$	•
6. $BE \rightarrow C$	• $(BE)^+ = ABCDEG$	• BE is a key
7. $CG \rightarrow BD$	• $(CG)^+ = ABCDEG$	• CG is a key
8. $CE \rightarrow AG$	• $(CE)^+ = ABCDEG$	• CE is a key

Applying Armstrong's Axioms gives BD and CD are also keys.

Generate minimal cover - Working

	Original *FDs*	single *RHS*	redundant *LHS*	minimal FDs
				Find AB^+ wrt the rest of FDs
1	$AB \to C$	$AB \to C$	$\boxed{AB \to C}$	$AB^+ = \{A, B\}$ $C \notin AB^+$ keep ①
2	$C \to A$	$C \to A$	$C \to A$	
3	$BC \to D$	$BC \to D$	$BC \to D$	
4	$ACD \to B$	$ACD \to B$	$\boxed{\cancel{A}CD \to B}$	$CD^+ = \{C, D, \overset{②}{A}, \overset{⑤}{E}, \overset{⑥}{G}, \underset{⑧}{B}\}$ $B \in CD^+$, remove ④ or ⑧
5	$D \to EG$	$D \to E$	$D \to E$	
6	$BE \to C$	$D \to G$	$D \to G$	
7	$CG \to BD$	$BE \to C$	$BE \to C$	
8	$CE \to AG$	$CG \to B$	$CG \to B$	
9		$CG \to D$	$CG \to D$	
10		$CE \to A$	$C\cancel{E} \to A$	
11		$CE \to G$	$CE \to G$	

Generate minimal cover

single *RHS*

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

redundant *LHS*

1. $AB \to C$
2. $C \to A$
3. $BC \to D$
4. $\cancel{A}CD \to B$
5. $D \to E$
6. $D \to G$
7. $BE \to C$
8. $CG \to B$
9. $CG \to D$
10. $C\cancel{E} \to A$
11. $CE \to G$

duplications

- delete ⑧ = ⑨ + ④
- delete ⑩ = ②

minimal FDs

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

The synthesis process

Combine same LHS	Form relations	The 3NF decomposition
1 $AB \to C$	1 $\{A,B,C\}$	1 $\{A,B,C\}$
2 $C \to A$	2 $\{C,A\}$	2 ② is a proper subset of ①
3 $BC \to D$	3 $\{B,C,D\}$	3 $\{B,C,D\}$
4 $CD \to B$	4 $\{C,D,B\}$	4 ④ is a subset of ③
5 $D \to E, D \to G$	5 $\{D,E,G\}$	5 $\{D,E,G\}$
6 $BE \to C$	6 $\{B,E,C\}$	6 $\{B,E,C\}$
7 $CG \to D$	7 $\{C,G,D\}$	7 $\{C,G,D\}$
8 $CE \to G$	8 $\{C,E,G\}$	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 - Lossless Join Property

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:

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

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”.

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 ith row, if the jth attribute is in R_i, cell (i, j) should contain value a_j, otherwise, b_{ij}.

	A	B	C	D	E	G
$R_1\{A, B, C\}$	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
$R_2\{B, C, D\}$	b_{21}	a_2	a_3	a_4	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

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:

A	B	C	D	E	G
a_1	a_2	a_3	a_4	a_5	a_6

successfully reconstructed

	A	B	C	D	E	G
$R_1\{A,B,C\}$	a_1	a_2	a_3	a_4	a_5	a_6
$R_2\{B,C,D\}$	a_1	a_2	a_3	a_4	a_5	a_6
$R_3\{D,E,G\}$				a_4	a_5	a_6
$R_4\{B,E,C\}$	a_1	a_2	a_3	a_4	a_5	a_6
$R_5\{C,G,D\}$	a_1	a_2	a_3	a_4	a_5	a_6
$R_6\{C,E,G\}$	a_1		a_3		a_5	a_6

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

Manual normalization

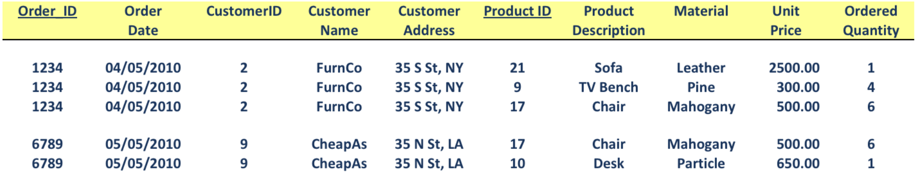
Data typically employed in a business is usually presented as below.

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

However to be useful in a database the data needs to be normalized.

First normal form – 1NF

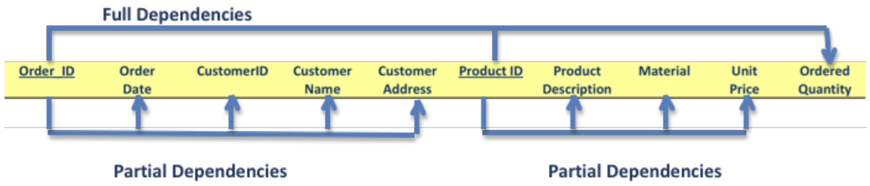
First normal form (1NF) requires attribute values be *atomic* — that is, individual values rather than lists or sets and that a primary key is defined.



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

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

Second normal form – 2NF



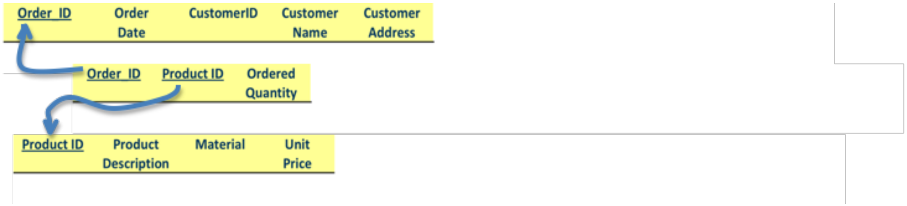
A relation in first normal form (1NF) has the above functional dependencies.

A relation is in Second normal for (2NF) if it is in 1NF and contains **no** partial dependencies.

A *partial functional dependency* exists when a *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.

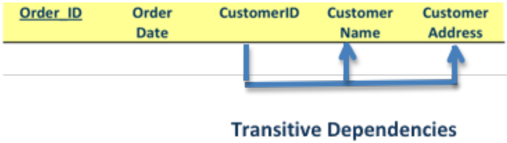


Order_ID	Order Date	CustomerID	Customer Name	Customer Address

Order_ID	Product ID	Ordered Quantity

Product ID	Product Description	Material	Unit Price

Third normal form – 3NF



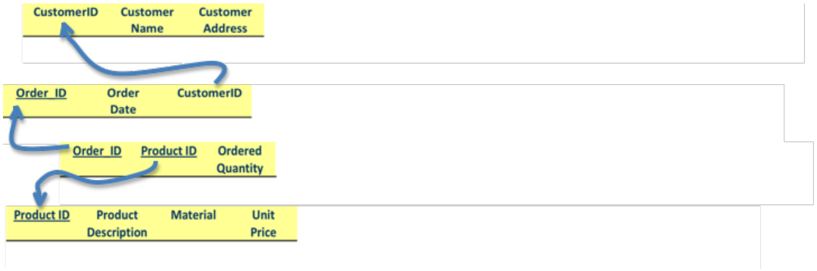
A relation in second normal form (2NF) still has the above functional dependencies.

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 *nonkey* attributes.

Third normal form – 3NF

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



Practical Use of Normalization

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

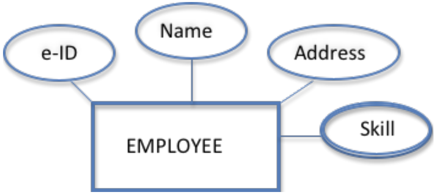
When the ER diagram is converted into tables, any functional dependencies should be identified and a decision made as to whether the tables should be normalized or not.

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

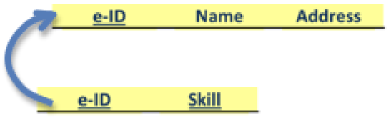
A *Single-valued* attribute of an entity instance has just one value. This has been the typical scenario we have discussed so far and know how this is handled in a schema definition.

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.



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.