6. Relational Database Design – 3NF - BCNF
Looking for a “Good” Form

• Recall that the goal of a good database design are
  – Lossless decomposition - necessary in order to ensure correctness of the data
  – Dependency preservation – not necessary, but desirable in order to achieve efficiency of updates
  – Good form – desirable in order to avoid redundancy.

• But what it means for a table to be in good form?

• If the domains of all attributes in a table contain only atomic values, then the table is in First Normal Form (1NF).
• In other words, there are no nested tables, multi-valued attributes, or complex structures such as lists.
• Relational tables are always in 1NF, according to the definition of the relational model.
Third Normal Form (3NF)

- R is a relation schema, with the set F of FDs
- R is in 3NF if and only if
  - for each FD: $X \rightarrow \{A\}$ in $F^+$
- Then
  - $A \in X$ (trivial FD), or
  - X is a superkey for R, or
  - A is prime attribute for R
- In words: For every FD that does not contain extraneous (useless) attributes:
  - the LHS is a candidate key, or
  - the RHS is a prime attribute, i.e., it is an attribute that is part of a candidate key
3NF Example

- \( R = (B, C, E) \)
  \[ F = \{ \{E\} \rightarrow \{B\}, \{B, C\} \rightarrow \{E\} \} \]

- Remember that you always have to find all candidate keys in order to determine the normal form of a table
- Two candidate keys: \( BC \) and \( EC \)
  \( \{E\} \rightarrow \{B\} \) B is prime attribute
  \( \{B, C\} \rightarrow \{E\} \) BC is a candidate key
- None of the FDs violates the rules of the previous slide. Therefore, \( R \) is in 3NF
Redundancy in 3NF

- Bank-schema = (Branch B, Customer C, Employee E)
- \(F = \{\{E\} \rightarrow \{B\}\}, \) e.g., an employee works in a single branch
- \(\{B,C\} \rightarrow \{E\}\}, \) e.g., when a customer goes to a certain branch s/he is always served by the same employee

<table>
<thead>
<tr>
<th>Branch</th>
<th>Customer</th>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKUST</td>
<td>Wong</td>
<td>Au</td>
</tr>
<tr>
<td>HKUST</td>
<td>Chin</td>
<td>Au</td>
</tr>
<tr>
<td>Central</td>
<td>Wong</td>
<td>Jones</td>
</tr>
<tr>
<td>Central</td>
<td>null</td>
<td>Cheng</td>
</tr>
</tbody>
</table>

- A 3NF table still has problems
  - redundancy (e.g., we repeat that Au works at HKUST branch)
  - need to use null values (e.g., to represent that Cheng works at Central even though he is not assigned any customers).
Algorithm for 3NF Synthesis

Let R be the initial table with FDs F
Compute the canonical cover $F_c$ of F
$S=\emptyset$

for each FD $X \rightarrow Y$ in the canonical cover $F_c$
    If none of the tables contains $X,Y$
        $S=S \cup (X,Y)$

if none of the tables contains a candidate key for R
    Choose any candidate key CN
    $S=S \cup$ table with attributes of CN

The algorithm always creates a lossless-join, dependency-preserving, 3NF decomposition.
3NF Example

- Bank = (branch-name, customer-name, banker-name, office-number)
- Functional dependencies (also canonical cover):
  \( \{\text{banker-name}\} \rightarrow \{\text{branch-name}, \text{office-number}\} \)
  \( \{\text{customer-name, branch-name}\} \rightarrow \{\text{banker-name}\} \)
- Candidate Keys: \{\text{customer-name, branch-name}\} or \{\text{customer-name, banker-name}\}
- \{\text{banker-name}\} \rightarrow \{\text{office-number}\} violates 3NF
- 3NF tables – for each FD in the canonical cover create a table
  Banker = (banker-name, branch-name, office-number)
  Customer-Branch = (customer-name, branch-name, banker-name)
- Since Customer-Branch contains a candidate key for Bank, we are done.
- Question: is the decomposition lossless and dependency preserving?
  Answer: Yes – all decompositions generated by this algorithm have these properties
Boyece-Codd Normal Form (BCNF)

- R is a relation schema, with the set F of FDs
- R is in BCNF if and only if
  for each FD: $X \rightarrow \{A\}$ in $F^+$
- Then
  $A \in X$ (trivial FD), or
  $X$ is a superkey for R

- **In words:** For every FD that does not contain extraneous (useless) attributes, the LHS of every FD is a candidate key.
- BCNF tables have no redundancy.
- If a table is in BCNF it is also in 3NF (and 2NF and 1NF)
BCNF Example

- \( R = \{B, C, E\} \)
  \[ F = \{\{E\} \rightarrow \{B\}, \{B, C\} \rightarrow \{E\}\} \]
- Two candidate keys: BC and EC
  \{B, C\} \rightarrow \{E\} does not violate BCNF because BC is a key
  \{E\} \rightarrow \{B\} violates BCNF because E is not a key
- In order to achieve BCNF we have to decompose the table but how?
  Since the decomposition must be lossless, we only have one option: \( R_1(B, E) \), and \( R_2(C, E) \). The common attribute E should be key of one fragment, here \( R_1 \).
BCNF Example (cont)

- Bank-schema = (Branch B, Customer C, Employee E)
- \( F = \{\{E\} \rightarrow \{B\}, \{B,C\} \rightarrow \{E\}\} \)
- Decompose into \( R_1(B,E) \), and \( R_2(C,E) \)

<table>
<thead>
<tr>
<th>Branch</th>
<th>Customer</th>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKUST</td>
<td>Wong</td>
<td>Au</td>
</tr>
<tr>
<td>HKUST</td>
<td>Chin</td>
<td>Au</td>
</tr>
<tr>
<td>Central</td>
<td>Wong</td>
<td>Jones</td>
</tr>
<tr>
<td>Central</td>
<td>null</td>
<td>Cheng</td>
</tr>
</tbody>
</table>

- We have avoided the problems of redundancy and null values of 3NF
We can generate the original table by joining the two fragments, using an *outer join*

<table>
<thead>
<tr>
<th>Branch</th>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKUST</td>
<td>Au</td>
</tr>
<tr>
<td>Central</td>
<td>Jones</td>
</tr>
<tr>
<td>Central</td>
<td>Cheng</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wong</td>
<td>Au</td>
</tr>
<tr>
<td>Chin</td>
<td>Au</td>
</tr>
<tr>
<td>Wong</td>
<td>Jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch</th>
<th>Cust.</th>
<th>Empl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKUST</td>
<td>Wong</td>
<td>Au</td>
</tr>
<tr>
<td>HKUST</td>
<td>Chin</td>
<td>Au</td>
</tr>
<tr>
<td>Central</td>
<td>Wong</td>
<td>Jones</td>
</tr>
<tr>
<td>Central</td>
<td>null</td>
<td>Cheng</td>
</tr>
</tbody>
</table>

- Is the decomposition dependency preserving?
  - No. We lose \(\{B,C\} \rightarrow \{E\}\)

- Can we have a dependency preserving decomposition?
  - No. No matter how we break we lose \(\{B,C\} \rightarrow \{E\}\) since it involves all attributes

11
Observations about BCNF

• Best Normal Form
• Avoids the problems of redundancy and all anomalies
• There is always a lossless decomposition that generates BCNF tables
• However, we may not be able to preserve all dependencies
• Next step: an algorithm for automatically generating BCNF tables.
Algorithm for BCNF Decomposition

Let R be the initial table with FDs F
S={R}
Until all relation schemes in S are in BCNF
    for each R in S
        for each FD X → Y that violates BCNF for R
            S = (S – {R}) ∪ (R-Y) ∪ (X,Y)
    enduntil

• This is a simplified version. In words:
• When we find a table R with BCNF violation X→Y we:
  1] Remove R from S
  2] Add a table that has the same attributes as R except for Y
  3] Add a second table that contains the attributes in X and Y
Let us consider the relation scheme $R=(A,B,C,D,E)$ and the FDs:
\[
\{A\} \rightarrow \{B,E\}, \ {C} \rightarrow \{D\}
\]
- Candidate key: AC
- Both functional dependencies violate BCNF because the LHS is not a candidate key
- Pick \(\{A\} \rightarrow \{B,E\}\)
- We can also choose \(\{C\} \rightarrow \{D\}\) – different choices lead to different decompositions.
- \((A,B,C,D,E)\) generates $R_1=(A,C,D)$ and $R_2=(A,B,E)$
- Do we need to decompose further?
BCNF Decomposition Example (cont)

- \((A,C,D)\) and \((A,B,E)\)
- \{A\} \rightarrow \{B,E\}, \{C\} \rightarrow \{D\}
- We need to decompose \(R1 = (A,C,D)\) because of the FD \(\{C\} \rightarrow \{D\}\)
- Thus \((A,C,D)\) is replaced with \(R3 = (A,C)\) and \(R4 = (C,D)\).
- Final decomposition: \(R2 = (A,B,E)\), \(R3 = (A,C)\), \(R4 = (C,D)\)

- Is the decomposition lossless?
  - Yes the algorithm **always** creates lossless decompositions. In step \(S = (S - \{R\}) \cup (R-Y) \cup (X,Y)\) we replace \(R\) with tables \((R-Y)\) and \((X,Y)\) that have \(X\) as the common attribute and \(X \rightarrow Y\), i.e., \(X\) is the key of \((X,Y)\)

- Is the decomposition dependency preserving?
  - Yes because \(F2 = \{\{A\} \rightarrow \{B,E\}\}, F3 = \emptyset, F4 = \{\{C\} \rightarrow \{D\}\}\) and \((F2 \cup F3 \cup F4)^+ = F^+\)
  - **But remember:** sometimes we may not be able to preserve dependencies
Testing if a FD violates BCNF

- **Important question:** which dependencies to check for BCNF violations? $F$ or $F^+$?
- **Answer-Part 1:** To check if a table $R$ with a given set of FDs $F$ is in BCNF, it suffices to check only the dependencies in $F$
- **Consider** $R (A, B, C, D)$, with $F = \{\{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}\}$
  - The key is $\{A,D\}$.
  - $R$ violates BCNF because the LHS of both $\{A\} \rightarrow \{B\}$ and $\{B\} \rightarrow \{C\}$. Neither $A$ nor $B$ is a key.
  - We can see that by simply using $F$ - we do not need $F^+$ (e.g., we do not need to check the implicit FD $\{A\} \rightarrow \{C\}$)
- **We can show** that if none of the dependencies in $F$ causes a violation of BCNF, then none of the dependencies in $F^+$ will cause a violation of BCNF either.
Answer-Part 2: However, using only F is insufficient when testing a fragment in the decomposition of R

Consider again R(A,B,C,D), with $F = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\} \}$ that violates BCNF

- Decompose R into and $R_1(A,C,D)$ and $R_2(A,B)$
- There is no FD in F that contains only attributes from $R_1(A,C,D)$ so we might be mislead into thinking that $R_1$ is in BCNF.
- In fact, dependency $\{A\} \rightarrow \{C\}$ in $F^+$ shows that $R_1$ is not in BCNF.
- Therefore, for the decomposed relations we also need to consider dependencies in $F^+$
Different BCNF Decompositions

- The different possible orders in which we consider FDs violating BCNF in the algorithm may lead to different decompositions.
- Previous example: $R(A,B,C,D)$, $F = \{\{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}\}$
- Previous BCNF decomposition: $R_2(A,B)$, $R_3(A,D)$, $R_4(A,C)$
- Question: is the decomposition dependency preserving?

  Answer: No – we lost the dependency $\{B\} \rightarrow \{C\}$
- Question: Can you obtain a dependency preserving decomposition?
- Answer: Yes – in the first decomposition we first applied violation $\{A\} \rightarrow \{B\}$. If, instead, we apply $\{B\} \rightarrow \{C\}$ we obtain:
  - $R_1 = (A,B,D)$ and $R_2 = (B,C)$
  - We decompose $R_1 = (A,B,D)$ further using $\{A\} \rightarrow \{B\}$ to obtain:
    - $R_3 = (A,D)$ and $R_4 = (A,B)$
  - The final decomposition $R_2 = (B,C)$, $R_3 = (A,D)$, $R_4 = (A,B)$ is dependency preserving.
Normalization Goals

• Goal for a relational database design is:
  – BCNF.
  – Lossless join.
  – Dependency preservation.

• If we cannot achieve this, we accept one of
  – Lack of dependency preservation in BCNF
  – Redundancy due to use of 3NF
When an E-R diagram is carefully designed, the tables generated from the E-R diagram should not need further normalization.

However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity.

E.g. employee entity with attributes department-number and department-address, and an FD department-number → department-address.

- Good design would have made department an entity.
Universal Relation Approach

- We start with a single universal relation and we decompose it using the FDs (no ER diagrams)
- Assume Loans(branch-name, loan-number, amount, customer-id, customer-name) and FDs:
  - \{loan-number\} → \{branch-name, amount, customer-id\}
  - \{customer-id\} → \{customer-name\}
- We apply existing decomposition algorithms to generate tables:
  - Loan(loan-number, branch-name, amount, customer-id)
  - Customer(customer-id, customer-name)
Denormalization for Performance

• May want to use non-normalized schema for performance
• E.g. displaying customer-name along with loan-number and amount requires join of loan with customer

• Alternative 1: Use denormalized relation containing attributes of loan as well as customer with all above attributes
  – faster lookup
  – Extra space and extra execution time for updates
  – extra coding work for programmer and possibility of error in extra code

• Alternative 2: use a materialized view defined as loan JOIN customer
  – Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors
Other Design Issues

• Some aspects of database design are not caught by normalization

• Examples of bad database design, to be avoided:

• Instead of \textit{earnings}(\textit{company-id, year, amount}), use


  • Above are in BCNF, but make querying across years difficult and needs new table each year


  • Also in BCNF, but also makes querying across years difficult and requires new attribute each year.