Normal Forms. BCNF and 3NF Decompositions
Review of Functional Dependencies (FDs)

- A **functional dependency** \( X \rightarrow Y \) holds over relation \( R \) if for every instance \( r \) of \( R \)
  - \( t_1, t_2 \in r, \pi_X(t_1) = \pi_X(t_2) \) implies \( \pi_Y(t_1) = \pi_Y(t_2) \)
  - given two tuples in \( r \), if the \( X \) values agree, \( Y \) values must also agree
- We say that the attributes \( X \) “determine” the attributes \( Y \), or the attribute of \( Y \) "depend upon" \( X \)
- \( X \) and \( Y \) are sets of attributes (we use other letters for individual attributes)
- **FD** is a statement about *all* allowable relations.
  - Identified based on semantics of application (business logic)
  - Given an instance \( r \) of \( R \), we can check if it violates some FD \( f \), but we cannot tell if \( f \) holds over \( R \)!
From last time: Sample Relation

Hourly_Emps (ssn, name, lot, rating, wage, hrs_worked)

- Denote relation schema by attribute initials: SNLRWH
- Constraints (functional dependencies, or FDs for short)
  - **ssn** is the key: \( S \rightarrow SNLRWH \)
  - **rating** determines **wage**: \( R \rightarrow W \)
    - E.g., worker with rating 10 receives 20$/hr

- Here \( X \rightarrow Y \) means \( Y \) depends on \( X \): if 2 rows agree on \( X \), they also agree on \( Y \), or equivalently, disagree on \( Y \Rightarrow \) disagree on \( X \)
  - It’s always true that key \( \rightarrow \) other cols
  - Here also \( R \rightarrow W \): agree on \( R \) means agree on \( W \). This is a non-key-related FD, one that indicates we need to do a "decomposition"
Using the non-key FD to guide decomposition

Create 2 smaller tables!

Hourly_Emps2

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

Wages

<table>
<thead>
<tr>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

- The FD $R \rightarrow W$ now lies in the Wages table, defining its key.
- The FD $S \rightarrow SNLRWH$ simplifies to $S \rightarrow SNLRH$, defining the key for Hourly_Emps2.
- The join of these two tables yields the original table.
Attribute Closure for a similar table, abbreviated notation for it

\[ X^+ = X \]

Repeat

\[ Y = X^+ \] (remember starting \( X^+ \) value)

Find FDs in \( F \) with LHSs completely included in \( X^+ \)

Add RHS of each such FD to \( X^+ \)

Until \( Y = X^+ \) (same as at start of this pass)

Relation Hourly(ssn, rating, wage)

Given (1) \( S \rightarrow R \), (2) \( R \rightarrow W \) (here \( R \) is rating, not whole relation)

\( S^+ = S \), look for FDs with LHS in \( S \), find \( S \rightarrow R \)

\( S^+ = SR \), look for FDs with LHS in \( SR \), find \( R \rightarrow W \)

\( S^+ = SRW = \text{Relation} \), so \( S \) is a superkey, and a key

Abbreviated notation: OK for homework:

\( S^+ = S = SR \) by (1), = \( SRW \) by (2), = Relation, so \( S \) is a superkey and a key

\( R^+ = R = RW \) by (2) \quad \text{W+ = W, done} \)
Checking Functional Dependencies (FDs)

Figure 19.3 An Instance that Satisfies $AB \rightarrow C$

- Check FD $AB \rightarrow C$ on this instance:
  - Look for rows that are same on A and B:
  - First two, and for these, C values are the same
  - No more pairs of rows are the same on A and B,
  - So $AB \rightarrow C$ holds for this instance
From Last time: Example of FD Inference

- Pg. 613: a contract with id C is an agreement that a supplier S will supply Q items of part P to project J associated with department D: the value of this contract is V

- **Contracts**\( (cid, sid, jid, did, pid, qty, value) \), and:
  - Contract id, supplier, project, department, part
  - C is the key: \( C \rightarrow CSJDPQV \)
  - Project purchases each part using single contract: \( JP \rightarrow C \)
    - A certain project J and part P purchased determines a particular contract C
  - Dept purchases at most one part from a supplier: \( SD \rightarrow P \)
    - A certain department D and supplier determine a certain part P, if any
An Example of FD Inference

- Contracts($cid, sid, jid, did, pid, qty, value$), and:
  - Contract id, supplier, project, department, part
  - $C$ is the key: $C \rightarrow CSJDPQV$
  - Project purchases each part using single contract: $JP \rightarrow C$
  - Dept purchases at most one part from a supplier: $SD \rightarrow P$

- $JP \rightarrow C, C \rightarrow CSJDPQV \implies JP \rightarrow CSJDPQV$
- $SD \rightarrow P \implies SDJ \rightarrow JP$
- $SDJ \rightarrow JP, JP \rightarrow CSJDPQV \implies SDJ \rightarrow CSJDPQV$

• So SDJ is a superkey for the table. Here we used transitivity, then augmentation and set definition, then transitivity again.
• **But we want to avoid this low-level math work: we can use attribute closure instead...**
Same result using attribute closure

- **Contracts** *(cid, sid, jid, did, pid, qty, value)*, and:
  - Contract id, supplier, project, department, part
  - C is the key: (1) \( C \rightarrow CSJDPQV \)
  - Project purchases each part using single contract: (2) \( JP \rightarrow C \)
  - Dept purchases at most one part from a supplier: (3) \( SD \rightarrow P \)

To show SDJ is a superkey for the table, using attribute closure, with abbreviated notation:

\[
SDJ^+ = SDJ, \quad SDJP \text{ by (3), } SDJPC \text{ by (2), } SDJPCR \text{ by (1), } R \text{ (all attributes)}
\]

So SDJ is a superkey of Contracts

To show a key: need to check subsets SD+, DJ+, SJ+
Decomposition of a Relation Schema

- A decomposition of R replaces it by two or more relations
  - Each new relation schema contains a subset of the attributes of R
  - Every attribute of R appears in one of the new relations
  - E.g., SNLRWH decomposed into SNLRH and RW

- Decompositions should be used only when needed
  - Cost of join will be incurred at query time

- Problems may arise with (improper) decompositions
  - Reconstruction of initial relation may not be possible
  - Dependencies cannot be checked on smaller tables
Lossless Join Decompositions

Decomposition of R into X and Y is lossless-join if:

- \( \pi_X (r) \bowtie \pi_Y (r) = r \)

It is always true that \( r \subseteq \pi_X (r) \bowtie \pi_Y (r) \)

- In general, the other direction does not hold!
  - i.e., the join has “extra” rows, not in r
- If the other direction holds, the decomposition is lossless-join.

It is essential that all decompositions used to deal with redundancy be lossless!
Incorrect Decomposition

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Natural Join

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Condition for Lossless-join

- The decomposition of $R$ into $X$ and $Y$ is lossless-join wrt $F$ if and only if the closure of $F$ contains:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$

- Note that $X \cap Y$ constitutes the join columns, so the join columns need to be a superkey of $X$ or $Y$.

- In particular, the decomposition of $R$ into $UV$ and $R \cdot V$ is lossless-join if $U \rightarrow V$ holds over $R$.
  - Example: $SNLRWH$, $R \rightarrow W$ decomposed into $SNLRH$ and $RW$
Consider CSJDPQV, C is key, JP \rightarrow C and SD \rightarrow P.

Consider decomposition: CSJDPQV and SDP

Problem: Checking JP \rightarrow C requires a join!

We want J, P, and C all in one table of the decomposition

Then we'll be able to say we've "preserved" this dependency

Dependency preserving decomposition (Intuitive):

If R is decomposed into X and Y, and we enforce the FDs that hold on X, Y then all FDs that were given to hold on R must also hold

We get a dependency preserving decomposition if all the FDs lie in different tables of the decomposition, and in some additional cases that require a lot of machinery (math-like) that we don't have time to study.
Dependency Preserving Decompositions

- Dependency preserving does not imply lossless join:
  - ABC, \( A \rightarrow B \), decomposed into AB and BC. Not lossless.
  - ABC should be decomposed into AB and AC, which provide a lossless join.
Normal Forms

- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized.
- Recall from the intro that 3NF is our goal for applications.
- Role of FDs in detecting redundancy:
  - Consider a relation $R$ with attributes $AB$ (and others too)
    - **No FDs hold**: There is no redundancy
    - **Given $A \rightarrow B$**:
      - Several tuples could have the same $A$ value
      - If so, they’ll all have the same $B$ value!
      - This is a potential redundancy
      - **Cure**: decomposition into two tables
Boyce-Codd Normal Form (BCNF)

- Relation R with FDs F is in BCNF if, for all $X \rightarrow A$ in $F^+$
  - $A \subseteq X$ (called a trivial FD), or
  - $X$ is a superkey of R.

- The only non-trivial FDs allowed are key constraints

- Every non-key column must depend on the whole key (2NF)
  - Because if non-key B depended on part of the key, say $X'$, we would have FD $X' \rightarrow B$, where B is not in $X'$ and $X'$ is not a superkey, so the table is not BCNF.

- Note that BCNF is the same as 3NF unless there are multiple keys and at least one multicolumn key, so very close to our goal.
Boyce-Codd Normal Form (BCNF)

- Relation R with FDs F is in **BCNF** if, for all $X \rightarrow A$ in $F^+$
  - $A \subseteq X$ (called a *trivial* FD), or
  - $X$ is a superkey of R.

- In a BCNF relation, every non-key attribute is determined by the whole key under consideration
- No redundancy (provable by FDs), no update, insertion, or deletion anomalies are possible
What FDs violate BCNF?

X -> A violates BCNF (for a relation) if

- X is a proper subset of some key K (partial dependency)
  - Example: Reserves SBDC, only key SBD, FD S->C
    \[\text{SBDC}\]   (this also violates 2NF)

- X is not a proper subset of any key (transitive dependency)
  - Example: Hourly_emps, SNLRWH, only key S, FD R->W *
    \[\text{SNLRWH}\]   (this example is 2NF)

- The target of the transitive FD can be \text{inside} the key

*If R is also a key, this would be BCNF
What FDs violate BCNF?

Figure 19.7 Partial Dependencies

Case 1: A not in KEY
(this violates 2NF)

Case 1: A not in KEY
(this violates 3NF and BCNF)

Case 2: A is in KEY
(this violates BCNF, OK for 3NF)

Figure 19.8 Transitive Dependencies
Decomposition into BCNF (and 3NF too)

- Consider relation $R$ with FDs $F$. If $X \rightarrow Y$ violates BCNF*, decompose $R$ into $R - Y$ and $XY$.

- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.

- e.g., CSJDPQV, key C, JP $\rightarrow$ C, SD $\rightarrow$ P, J $\rightarrow$ S

- To deal with SD $\rightarrow$ P, decompose into SDP, CSJDQV.

- To deal with J $\rightarrow$ S, decompose CSJDQV into JS and CJDQV

- *or 3NF, defined soon.
In general, several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!
BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF
  - e.g., ABC, AB → C, C → A
  - Can’t decompose while preserving first FD; not in BCNF
  - We will see that ABC is in 3NF.
Third Normal Form (3NF)

- Relation $R$ with FDs $F$ is in 3NF if, for all $X \rightarrow A$ in $F^+$
  - $A \in X$ (called a trivial FD), or
  - $X$ contains a key for $R$, or
  - $A$ is part of some key for $R$ (Here, $A$ is a single attribute)

- *Minimality* of a key is crucial in third condition above!

- If $R$ is in BCNF, it is also in 3NF.

- If $R$ is in 3NF, some redundancy is possible
  - compromise used when BCNF not achievable

- Lossless-join, dependency-preserving decomposition of $R$ into a collection of 3NF relations always possible.

- 2NF: replace "$X$ contains a key" with "$X$ contains a key or is determined by a key" to allow transitive dependencies
Decomposition into 3NF

- Lossless join decomposition algorithm also applies to 3NF
- To ensure dependency preservation, one idea:
  - If \( X \rightarrow Y \) is not preserved, add relation \( XY \)
  - Refinement: Instead of the given set of FDs \( F \), use a minimal cover for \( F \)
- Example: \( CSJDPQV, JP \rightarrow C, SD \rightarrow P, J \rightarrow S \)
  - Choose \( SD \rightarrow P \), result is \( SDP \) and \( CSJDPQV \)
  - Choose \( J \rightarrow S \), result is \( JS \) and \( CJDQV \), all 3NF
  - Add \( CJP \) relation
Summary of Schema Refinement

- **BCNF**: relation is free of FD redundancies
  - Having only BCNF relations is desirable
  - If relation is not in BCNF, it can be decomposed to BCNF
    - Lossless join property guaranteed
    - But some FD may be lost (i.e., dependency may not be preserved)
- **3NF** is a relaxation of BCNF
  - Guarantees both lossless join and FD preservation
- Decompositions may lead to performance loss
  - *performance requirements* must be considered when using decomposition
What Does 3NF Achieve?

- If relation is in 3NF, redundancy may still occur
  - e.g., Reserves $\text{SBDC, S} \rightarrow \text{C, C} \rightarrow \text{S}$ is in 3NF
  - Each sailor has a credit card $\text{C}$, and it’s unique to him/her
  - $\text{CBD}^+ = \text{CBDS}$, so CBD is a key as well as SBD
  - That makes $\text{C} \rightarrow \text{S}$ acceptable for 3NF
  - But for each reservation of sailor $\text{S}$, same ($\text{S, C}$) pair is stored
  - 3NF is a compromise relative to BCNF, allows some redundancy to achieve dependency preservation