### Data Redundancy

**Table:**

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- Application constraint: all sailors with the same rating have the same wage (R → W)
- Problems due to data redundancy?

### Problems due to Data Redundancy

- **Update anomaly:** Can we change W in just the first tuple of SNLRWH?
- **Insertion anomaly:** What if we want to insert an employee and don't know the hourly wage for his rating?
- **Deletion anomaly:** If we delete all employees with rating 5, we lose the information about the wage for rating 5!

### Solution?

### Relation Decomposition

**Table:**

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

**Table:**

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

**Table:**

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

### Modifying ER Diagram

**Diagram:**

- **Update anomaly:** Can we change W in just the first tuple of SNLRWH?
- **Insertion anomaly:** What if we want to insert an employee and don't know the hourly wage for his rating?
- **Deletion anomaly:** If we delete all employees with rating 5, we lose the information about the wage for rating 5!

**Solution?**
Normal Forms

- First question is to ask whether any schema refinement is needed
- If a relation is in a normal form (BCNF, 3NF etc.), certain anomalies are avoided/minimized
- If not, decompose relation to normal form
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
  - No FDs hold: There is no redundancy here.
  - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - If X → Y and X → Z, then X → YZ
  - Armstrong’s Axioms (X, Y, Z are sets of attributes):
    - Reflexivity: If X ⊆ Y, then X → Y
    - Augmentation: If X → Y, then XZ → YZ for any Z
    - Transitivity: If X → Y and Y → Z, then X → Z
- These are sound and complete inference rules for FDs!

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD X → Y is in the closure of a set of FDs F. An efficient check:
  - Compute attribute closure of X (denoted X⁺) wrt F:
    - There is a linear time algorithm to compute this.
  - Check if Y is in X⁺
- Does F = {A → B, B → C, C D → E} imply A → E? Equivalently, is E in A⁺?
  - Can be used to find keys!!!

Outline

- Functional Dependencies
- Decompositions
- Normal Forms
**Outline**

- Functional Dependencies
- Decompositions
- Normal Forms

**Decomposition of a Relation Scheme**

Suppose that relation R contains attributes A1 ... An. A decomposition of R consists of replacing R by two or more relations such that:
- Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
- Every attribute of R appears as an attribute of one of the new relations.

Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.

E.g., can decompose SNLRWH into SNLRH and RW.

**Example Decomposition**

Decompositions should be used only when needed.
- SNLRWH has FDs S \( \rightarrow \) SNLRWH and R \( \rightarrow \) W
- Data duplication due to second FD
- Will make this more precise during the definition of normal forms

Decompose to SNLRH and RW
- What should we be careful about?

**Problems with Decompositions**

There are three potential problems to consider:
- Some queries become more expensive.
  - e.g., How much did sailor Joe earn? (salary = W*H)
- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  - Fortunately, not in the SNLRWH example.
- Checking some dependencies may require joining the instances of the decomposed relations.
  - Fortunately, not in the SNLRWH example.

Tradeoff: Must consider these issues vs. redundancy.

**Lossless Join Decompositions**

Decomposition of R into X and Y is lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:
- \( \pi_X(r) \supset \pi_Y(r) = r \)

It is always true that \( r \subseteq \pi_X(r) \supseteq \pi_Y(r) \)
- In general, the other direction does not hold! If it does, the decomposition is lossless-join.

Definition extended to decomposition into 3 or more relations in a straightforward way.

It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem 2.)

**More on 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
- \( X \cap Y \rightarrow Y \)

In particular, the decomposition of R into UV and R - V is lossless-join if \( U \rightarrow V \) holds over R.
**Dependency Preserving Decomposition**

- Consider CSJDPQV, C is key, JP \(\rightarrow\) C and SD \(\rightarrow\) P.
  - Decomposition: CSJDQV and SDP
  - (Is it lossless join?)
  - Problem: Checking JP \(\rightarrow\) C requires a join!

**Dependency preserving decomposition (Intuitive):**

- If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. (Avoids Problem 13.1)

**Projection of set of FDs F:** If R is decomposed into X, ... projection of F onto X (denoted \(F_X\)) is the set of FDs U \(\rightarrow\) V in \(F^+\) (closure of F) such that U, V are in X.

**Dependency Preserving Decompositions (Contd.)**

- Decomposition of R into X and Y is dependency preserving if \((F_X \cup F_Y)^+ = F^+\)
  - i.e., if we consider only dependencies in the closure \(F^+\) that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in \(F^+\).
  - Important to consider \(F^+\), not F, in this definition:
    - ABC, A \(\rightarrow\) B, B \(\rightarrow\) C, C \(\rightarrow\) A, decomposed into AB and BC.
    - Is this dependency preserving? Is C \(\rightarrow\) A preserved?????

**Dependency preserving does not imply lossless join:**

- ABC, A \(\rightarrow\) B, decomposed into AB and BC.
  - And vice-versa! (Example?)

**Outline**

- Functional Dependencies
- Decompositions
- Normal Forms

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

- Reln R with FDs F is in BCNF if, for all X \(\rightarrow\) A in \(F^+\)
  - A \(\not\in\) X (called a trivial FD), or
  - X contains a key for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

**Decomposition into BCNF**

- 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
  - 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., CSZ, CS \(\rightarrow\) Z, Z \(\rightarrow\) C
  - Can’t decompose while preserving 1st FD; not in BCNF.
  - Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP \(\rightarrow\) C, SD \(\rightarrow\) P and J \(\rightarrow\) S).
    - However, it is a lossless join decomposition.
**Third Normal Form (3NF)**

- Reln R with FDs F is in 3NF if, for all 
  - A \(\in\) X (called a trivial FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- Minimality of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no "good" decomps, or performance considerations).
- Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.

**What Does 3NF Achieve?**

- If 3NF violated by X \(\rightarrow\) A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs K \(\rightarrow\) X \(\rightarrow\) A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
  - But: even if reln is in 3NF, these problems could arise.
    - e.g., Reserves 5BDC, S \(\rightarrow\) C, C \(\rightarrow\) S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

**Decomposition into 3NF**

- Obviously, the algorithm for lossless join decomps into BCNF can be used to obtain a lossless join decomps into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If X \(\rightarrow\) Y is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to ‘preserve’ JP \(\rightarrow\) C. What if we also have J \(\rightarrow\) C?
- Refinement: Instead of the given set of FDs F, use a minimal cover for F.

**Minimal Cover for a Set of FDs**

- **Minimal cover** G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and "as small as possible" in order to get the same closure as F.
- e.g., A \(\rightarrow\) B, ABCD \(\rightarrow\) E, EF \(\rightarrow\) GH, ACDF \(\rightarrow\) EG has the following minimal cover:
  - A \(\rightarrow\) B, ACD \(\rightarrow\) E, EF \(\rightarrow\) G and EF \(\rightarrow\) H

**Summary of Schema Refinement**

- BCNF implies free of redundancies due to FDs
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
- If a lossless-join, dependency preserving decomposition into BCNF is not possible, consider 3NF
- Decompositions should be carried out and/or re-examined keeping performance issues in mind