Why Schema Refinement?

- We have learnt the advantages of relational tables …
- … but how to decide on the relational schema?
- At one extreme, store everything in single table
  - Huge redundancy
  - Leads to anomalies!
- We need to break the information into several tables
  - How many tables, and with what structures?
  - Having too many tables can also cause problems
    - E.g., performance, difficulty in checking constraints

Sample Relation

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

- Denote relation schema by attribute initial: SNLRWH
- Constraints (dependencies)
  - **ssn is the key:** S → SNLRWH
  - **rating determines wage:** R → W
  - E.g., worker with rating A receives 20$/hr

Anomalies

- Problems due to R → W:
  - **Update anomaly:** Change value of W only in a tuple – dependency violation
  - **Insertion anomaly:** How to insert employee if we don’t know hourly wage for that rating?
  - **Deletion anomaly:** If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Removing Anomalies

**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>

Create 2 smaller tables!

Dealing with Redundancy

- **Redundancy** is at the root of redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular functional dependencies, can be used to identify redundancy
- Main refinement technique: **decomposition** (replacing ABCD with, say, AB and BCD, or ACD and ABD)
- Decomposition should be used judiciously:
  - Decomposition may sometimes affect performance. Why?
  - What problems (if any) does decomposition cause?
    - Incorrect data
    - Loss of dependencies
**Functional Dependencies (FDs)**
- A functional dependency \( X \rightarrow Y \) holds over relation \( R \) if for every instance \( r \) of \( R \), \( \forall t_1, t_2 \in r \), \( \pi_X(t_1) = \pi_X(t_2) \) implies \( \pi_Y(t_1) = \pi_Y(t_2) \).
- Given two tuples in \( r \), if the \( X \) values agree, \( Y \) values must also agree.
- 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 \( R \)!

**FDs and Keys**
- FDs are a generalization of keys.
- A key uniquely identifies all attribute values in a tuple.
- That is a particular case of FD …
- … but not all FDs must determine ALL attributes.
- \( K \) is a key for \( R \) means that \( K \rightarrow R \)
- However, \( K \rightarrow R \) does not require \( K \) to be minimal!
- \( K \) can be a superkey as well.

**Reasoning About FDs**
- Given FD set \( \mathcal{F} \), we can usually infer additional FDs:
  - \( \mathcal{F}^+ = \text{closure of } \mathcal{F} \) is the set of all FDs that are implied by \( \mathcal{F} \)
  - Armstrong's Axioms (\( X, Y, Z \) are sets of attributes):
    - Reflexivity: If \( Y \subseteq X \), then \( X \rightarrow Y \)
    - Augmentation: If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \) for any \( Z \)
    - Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)
- These are sound and complete inference rules for FDs!

**Reasoning About FDs (cont'd)**
- Additional rules
  - Not necessary, but helpful
- Union and decomposition (splitting)
  - \( X \rightarrow Y \) and \( X \rightarrow Z \Rightarrow X \rightarrow YZ \)
  - \( X \rightarrow YZ \Rightarrow X \rightarrow Y \) and \( X \rightarrow Z \)

**An Example of FD Inference**
- Contracts(\( cid, sid, jid, did, pid, qty, value \)), and:
  - Contract id, supplier, project, department, part
  - C is the key: \( C \rightarrow \text{CSJDPQV} \)
  - Project purchases each part using single contract: \( JP \rightarrow C \)
  - Dept purchases at most one part from a supplier: \( SD \rightarrow P \)
- \( JP \rightarrow C, C \rightarrow \text{CSJDPQV} \) imply \( JP \rightarrow \text{CSJDPQV} \)
- \( SD \rightarrow P \) implies \( SDJ \rightarrow JP \)
- \( SDJ \rightarrow JP, JP \rightarrow \text{CSJDPQV} \) imply \( SDJ \rightarrow \text{CSJDPQV} \)

**Attribute Closure**
- Attribute closure of \( X \) (denoted \( X^+ \)) wrt FD set \( \mathcal{F} \):
  - Set of all attributes \( A \) such that \( X \rightarrow A \) is in \( \mathcal{F}^+ \)
  - Set of all attributes that can be determined starting from attributes in \( X \) and using FDs in \( \mathcal{F} \)
- Apply split rule such that all FDs have single attr in RHS
  - \( X = X \)
  - \( X \rightarrow X^+ \)
  - Search all FDs in \( F \) with LHS completely included in \( X^+ \)
  - Add RHS of those FDs to \( X^+ \)
  - Until \( Y = X \)
Verifying if given FD in FD-set closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure is exponential in number of attributes!

- But if we just want to check if a given FD \( X \rightarrow Y \) is in the closure of a set of FDs \( F \):
  - Can be done efficiently without need to know \( F^+ \)
  - Compute \( X^+ \) wrt \( F \)
  - Check if \( Y \) is in \( X^+ \)

Verifying if attribute set is a key

- Key verification can also be done with attribute closure

- To verify if \( X \) is a key, two conditions needed:
  - \( X^+ = R \)
  - \( X \) is minimal

- How to test minimality
  - Removing an attribute from \( X \) results in \( X' \) such that \( X'^+ \neq R \)