Features of Good Relational Designs

- Suppose we combine instructor and department into in_dep, which represents the natural join on the relations instructor and department

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>12412</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>76566</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58383</td>
<td>Califieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>83621</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozzari</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33436</td>
<td>Gold</td>
<td>67000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
</tbody>
</table>

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)
Decomposition

- The only way to avoid the repetition-of-information problem in the `in_dep` schema is to decompose it into two schemas – instructor and `department` schemas.
- Not all decompositions are good. Suppose we decompose

\[ employee(\text{ID, name, street, city, salary}) \]

into

\[
\begin{align*}
\text{employee1} & (\text{ID, name}) \\
\text{employee2} & (\text{name, street, city, salary})
\end{align*}
\]

The problem arises when we have two employees with the same name

- The next slide shows how we lose information -- we cannot reconstruct the original `employee` relation -- and so, this is a **lossy decomposition**.
Lossless Decomposition

- Let $R$ be a relation schema and let $R_1$ and $R_2$ form a decomposition of $R$
  - That is $R = R_1 \cup R_2$
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing $R$ with the two relation schemas $R_1 \cup R_2$
- Formally,
  $$\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$
- And, conversely a decomposition is lossy if
  $$r \subseteq \prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$

Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$
  $$R_1 = (A, B) \quad R_2 = (B, C)$$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>$r$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  $\prod_{A,B}(r)$  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
</tr>
</tbody>
</table>

  $\prod_{B,C}(r)$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

$\prod_{A}(r) \bowtie \prod_{B}(r)$
Normalization Theory

- Decide whether a particular relation $R$ is in “good” form.
- In the case that a relation $R$ is not in “good” form, decompose it into set of relations $\{R_1, R_2, \ldots, R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID.
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.
Functional Dependencies

$X \rightarrow A = \text{assertion about a relation } R \text{ that whenever two tuples agree on all the attributes of } X, \text{ then they must also agree on attribute } A$

Why do we care?

Knowing functional dependencies provides a formal mechanism to divide up relations (normalization)

- Saves space
- Prevents storing data that violates dependencies

---

Functional Dependencies Definition

- Let $R$ be a relation schema $\alpha \subseteq R$ and $\beta \subseteq R$
- The functional dependency $\alpha \rightarrow \beta$
  holds on $R$ if and only if for any legal relations $r(R)$, whenever any two tuples $t_1$ and $t_2$ of $r$ agree on the attributes $\alpha$, they also agree on the attributes $\beta$. That is,

  $$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A, B)$ with the following instance of $r$.

  $\begin{array}{cc}
  1 & 4 \\
  1 & 5 \\
  3 & 7 \\
  \end{array}$

- On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does NOT hold,
Keys of Relations

K is a key for relation R if:

1. $K \rightarrow$ all attributes of $R$. (Uniqueness)
2. For no proper subset of $K$ is (1) true. (Minimality)
   - If $K$ at least satisfies (1), then $K$ is a superkey.

Conventions
- Pick one key; underline key attributes in the relation schema.
- $X$, etc., represent sets of attributes; $A$ etc., represent single attributes.
- No set formers in FD's, e.g., $ABC$ instead of \{A, B, C\}.

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  $in_{dep} (ID, name, salary, dept\_name, building, budget)$.

  We expect these functional dependencies to hold:

  \[ dept\_name \rightarrow building \]

  \[ ID \rightarrow building \]

  but would not expect the following to hold:

  \[ dept\_name \rightarrow salary \]
### Example

<table>
<thead>
<tr>
<th>Lastname</th>
<th>Firstname</th>
<th>Student ID</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td></td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Superkey</td>
</tr>
</tbody>
</table>

Note: There are alternate keys

- Keys are \{Lastname, Firstname\} and \{StudentID\}

---

### Use of Functional Dependencies

- We use functional dependencies to:
  - To test relations to see if they are legal under a given set of functional dependencies.
    - If a relation $r$ is legal under a set $F$ of functional dependencies, we say that $r$ satisfies $F$.
  - To specify constraints on the set of legal relations
    - We say that $F$ holds on $R$ if all legal relations on $R$ satisfy the set of functional dependencies $F$.

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of instructor may, by chance, satisfy $name \rightarrow ID$. 
Trivial Functional Dependencies

- A functional dependency is **trivial** if it is satisfied by all instances of a relation

- Example:
  - $ID, \text{name} \rightarrow ID$
  - $\text{name} \rightarrow \text{name}$

- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Who Determines Keys/FD’s?

- We could assert a key $K$.
  - Then the only FD’s asserted are that $K \rightarrow A$ for every attribute $A$.
  - No surprise: $K$ is then the only key for those FD’s, according to the formal definition of “key.”

- Or, we could assert some FD’s and *deduce* one or more keys by the formal definition.

- Rule of thumb: FD’s either come from keyness, many-1 relationship, or from physics.
  - *E.g.*, “no two courses can meet in the same room at the same time” yields $\text{room time} \rightarrow \text{course}$.
Functional Dependencies (FD’s) and Many-One Relationships

- Consider \( R(A_1, \ldots, A_n) \) and \( X \) is a key then \( X \rightarrow Y \) for any attributes \( Y \) in \( A_1, \ldots, A_n \) even if they overlap with \( X \). Why?
- Suppose \( R \) is used to represent a many \( \rightarrow \) one relationship:
  - \( E_1 \) entity set \( \rightarrow E_2 \) entity set
  - where \( X \) key for \( E_1 \), \( Y \) key for \( E_2 \),
  - Then, \( X \rightarrow Y \) holds,
  - And \( Y \rightarrow X \) does not hold unless the relationship is one-one.
- What about many-many relationships?

Inferring FD’s

And this is important because …
- When we talk about improving relational designs, we often need to ask “does this FD hold in this relation?”

Given FD’s \( X_1 \rightarrow A_1, X_2 \rightarrow A_2, \ldots, X_n \rightarrow A_n \), does FD \( Y \rightarrow B \) necessarily hold in the same relation?
- Start by assuming two tuples agree in \( Y \). Use given FD’s to infer other attributes on which they must agree. If \( B \) is among them, then yes, else no.