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>12121</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>45655</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>76766</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>Califiri</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>15451</td>
<td>Mozari</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33436</td>
<td>Gold</td>
<td>87000</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

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

  into

  \[ \text{employee1}(ID, \text{name}) \]
  \[ \text{employee2}(\text{name}, \text{street}, \text{city}, \text{salary}) \]

  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,
  $$\Pi_{R_1}(r) \times \Pi_{R_2}(r) = r$$
- And, conversely a decomposition is lossy if
  $$r \subset \Pi_{R_1}(r) \times \Pi_{R_2}(r) = r$$

Example of Lossless Decomposition

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

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

- $r = (\alpha, \beta)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</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>
</tbody>
</table>

- $\Pi_{A,B}(r)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\Pi_{A}(r) \times \Pi_{B}(r)$

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

$X \rightarrow A$ = assertion about a relation $R$ that whenever two tuples agree on all the attributes of $X$, 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 $r$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

- 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. \( K \rightarrow \text{all attributes of } R \). (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:

  \( \text{in\_dep (ID, name, salary, dept\_name, building, budget)} \).

We expect these functional dependencies to hold:

  \( \text{dept\_name} \rightarrow \text{building} \)
  \( \text{ID} \rightarrow \text{building} \)

but would not expect the following to hold:

  \( \text{dept\_name} \rightarrow \text{salary} \)
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>Superkey</td>
</tr>
<tr>
<td>(2 attributes)</td>
<td></td>
<td></td>
<td></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 \( \text{name} \rightarrow \text{ID} \).
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 \( \text{room} \rightarrow \text{time} \rightarrow \text{course} \).
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 → one relationship:
  \( E_1 \) entity set → \( E_2 \) 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.
Closure of a Set of Functional Dependencies

- Given a set $F$ set of functional dependencies, there are certain other functional dependencies that are logically implied by $F$.
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - etc.
- The set of all functional dependencies logically implied by $F$ is the closure of $F$.
- We denote the closure of $F$ by $F^+$.

Algorithm

Define $Y^+ = \text{closure of } Y$ = set of attributes functionally determined by $Y$:
- Basis: $Y^+ := Y$.
- Induction: If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add $A$ to $Y^+$.
- End when $Y^+$ cannot be changed.
Example

\[ A \rightarrow B, \ BC \rightarrow D. \]

- \( A^+ = AB. \)
- \( C^+ = C. \)
- \( (AC)^+ = ABCD. \)

Given Versus Implied FD’s

Typically, we state a few FD’s that are known to hold for a relation \( R. \)
- Other FD’s may follow logically from the given FD’s; these are implied FD’s.
- We are free to choose any basis for the FD’s of \( R \) – a set of FD’s that imply all the FD’s that hold for \( R. \)
Finding All Implied FD’s

Motivation: Suppose we have a relation $ABCD$ with some FD’s $F$. If we decide to decompose $ABCD$ into $ABC$ and $AD$, what are the FD’s for $ABC$, $AD$?

• Example: $F = AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. It looks like just $AB \rightarrow C$ holds in $ABC$, but in fact $C \rightarrow A$ follows from $F$ and applies to relation $ABC$.

• Problem is exponential in worst case.

Example

$F = AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. What FD’s follow?

• $A^+ = A$; $B^+ = B$ (nothing).
• $C^+ = ACD$ (add $C \rightarrow A$).
• $D^+ = AD$ (nothing new).
• $(AB)^+ = ABCD$ (add $AB \rightarrow D$; skip all supersets of $AB$).
• $(BC)^+ = ABCD$ (nothing new; skip all supersets of $BC$).
• $(BD)^+ = ABCD$ (add $BD \rightarrow C$; skip all supersets of $BD$).
• $(AC)^+ = ACD$; $(AD)^+ = AD$; $(CD)^+ = ACD$ (nothing new).
• $(ACD)^+ = ACD$ (nothing new).

• All other sets contain $AB$, $BC$, or $BD$, so skip.

• Thus, the only interesting FD’s that follow from $F$ are: $C \rightarrow A$, $AB \rightarrow D$, $BD \rightarrow C$. 
**Lossless Decomposition**

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations $r$ on schema $R$
  $$ r = \prod_{R_1} (r) \times \prod_{R_2} (r) $$
- A decomposition of $R$ into $R_1$ and $R_2$ is lossless decomposition if at least one of the following dependencies is in $F^+$:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies.

**Example**

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), \ R_2 = (B, C)$
  - Lossless decomposition:
    $$ R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC $$
- $R_1 = (A, B), \ R_2 = (A, C)$
  - Lossless decomposition:
    $$ R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB $$
- **Note:**
  - $B \rightarrow BC$ is a shorthand notation for
  - $B \rightarrow \{B, C\}$
Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly.
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low.
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Product.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT dependency preserving.

Dependency Preservation Example

- Consider a schema:

```sql
dept_advisor(s_ID, i_ID, department_name)
```

- With function dependencies:

```sql
i_ID \rightarrow dept_name
s_ID, dept_name \rightarrow i_ID
```

- In the above design we are forced to repeat the department name once for each time an instructor participates in a `dept_advisor` relationship.
- To fix this, we need to decompose `dept_advisor`
- Any decomposition will not include all the attributes in `s_ID, dept_name \rightarrow i_ID`
- Thus, the composition NOT be dependency preserving.
FDs: Armstrong’s Axioms

• Reflexivity:
  – If \( \{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\} \Rightarrow A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m \)
  – Also called “trivial FDs”

• Augmentation:
  – \( A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m \Rightarrow A_1 A_2 \cdots A_n C_1 C_2 \cdots C_k \rightarrow B_1 B_2 \cdots B_m C_1 C_2 \cdots C_k \)

• Transitivity:
  – \( A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m \) and \( B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k \Rightarrow A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k \)