Schema Refinement & Normalization Theory: Functional Dependencies
Background

❖ We started with schema design
  - ER model; translation into a relational schema

❖ Then we studied relational query languages
  - relational algebra, relational calculus, SQL

❖ Now, we are back to schema design...
  - we will finish the semester by considering how to refine an existing schema

❖ But first: an aside on causality ...
Major Categories of Databases

- **Reference / Archival**
  - Example: university course catalog, splunk logs

- **Transactional**
  - Example: bank transactions

- **Analytic**
  - Example: purchasing patterns data warehouse

❖ Each has a distinct “personality”
❖ Can overlap! (e.g., transactional db can act as a reference model between transactions)
Major Categories of Databases

❖ **Reference**
- Stable - Detailed writes, updates reflect state of world. Reads seek knowledge.
- Examples: GMU course catalog, Wikipedia, design

❖ **Transactional**
- Dynamic - reads & writes bundled precisely in transactions, each leaving db in ~consistent state
- Examples: stock exchange, inventory

❖ **Analytic**
- “ETL” - Continual ingest / integration from diverse sources. Reads extract data for analytic models.
- Examples: consumer behavior, aviation safety, most “big data” systems
What is Causality?

❖ When some state of the world is an effect, determined by something else (the cause)
  - Data level: *what determines the price of a car?*, $y=f(x)$ where $x$ and $y$ are data fields
  - Conceptual level: *what determines government stability?* What determines inflation?
  - Philosophical / moral level: *what determines good & evil?*

❖ Our focus in databases is at the data level.
  - *But we can’t forget the larger context: causality is an important subject; it comes up over & over!*
Causal Data Relationships Across Database Categories

❖ In Reference / Archival databases
- causal relationships are descriptive (reflect declared policy or intrinsic causality, e.g., number_credits determines registration_time)
- Our focus in this lecture!

❖ In Transactional databases
- causal relationships are enforced (transaction consistency, e.g., credit_amt permits debit_amt)

❖ In Analytic databases
- causal relationships are predictive (features used in predictive models, e.g., snow_cover predicts accident_likelihood)
Insight

- Understanding causal data relationships within a database can help us better organize our data!

- An ER diagram may not (fully) acknowledge causality

- Now we will provide a methodology that does this explicitly!
An Example

- Consider the relation:
  Hourly_Emps(\textit{ssn, name, lot, rating, hrly\_wages, hrs\_worked})
  (call it SNLRWH, for short)

- What if we \textit{know} that an employee’s rating (R) determines their hrly\_wages (W)?

<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>

- The information in W is repeated (needlessly)!
Side Effect: Redundancy

- When part of database can be derived from other parts, *redundancy* often exists.
  - Example: the hrly_wage of Smiley is the same as the hrly_wage of Attishoo because they have the same rating and we know rating determines hrly_wage.

- This redundancy exists because of underlying integrity constraints called *functional dependencies*. The topic of this lecture!
  - rating -> hrly_wage
Is Redundancy Good or Bad?

❖ In this example, it seems like it would be cleaner to have a “lookup table” relating rating and hourly wages
  - Instead of awkwardly embedding such info in this table.

❖ Materialized views & data warehouses are highly redundant! But this saves expensive (re)computation in analytic ETL workflows.

❖ In both cases, it’s important to understand redundancy, its sources, & how to manage it.
Problems Which Can Arise From Redundancy

❖ **Update anomaly.** What if we change W in just the 1st tuple of SNLRWH?
❖ **Insertion anomaly.** What if we want to insert an employee and don’t know the hourly wage for his rating? Or use the wrong wage?
❖ **Deletion anomaly.** What if we delete all employees with rating 5, we lose the information about the wage for rating 5!
❖ **Sub-optimal design:** It would be a cleaner design to have a reference table relating ratings to wages.
❖ **Extra Storage:** (although storage is often cheap)
<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>
Overview

❖ We need to understand functional dependencies, and how they can lead to redundancy in a schema design.
  - update/insert/delete anomalies
  - intuitively awkward designs

❖ Schema refinement manages functional dependencies
  - Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).
  - Goal: decompose into a normal form with less redundancy
    • several are possible: e.g., 3NF, BCNF...

❖ Decomposition should be used judiciously; new issues can result:
  - Violate desirable properties of decompositions (lossless-join and dependency-preservation.)
  - Over-normalization
  - Decompositions are challenging in analytic databases, where functional dependencies can be unclear / being discovered.
Functional Dependencies (FDs)

- Functional dependencies are a (causal) type of integrity constraint:
  - The value of one set of attributes depends on the value of another set of attributes in the same relation

Example:

Emps (ssn, name, lot, rank, salary)

rank → salary (e.g., rank determines the value of salary)
rank → lot

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rank</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>Will Smith</td>
<td>A1</td>
<td>manager</td>
<td>150000</td>
</tr>
<tr>
<td>123456679</td>
<td>Bob Brown</td>
<td>A1</td>
<td>manager</td>
<td>150000</td>
</tr>
<tr>
<td>123456780</td>
<td>Ned Wang</td>
<td>A2</td>
<td>programmer</td>
<td>70000</td>
</tr>
<tr>
<td>092873817</td>
<td>Joyce Braddock</td>
<td>A1</td>
<td>analyst</td>
<td>85000</td>
</tr>
<tr>
<td>73681721</td>
<td>Edward Madison</td>
<td>A2</td>
<td>programmer</td>
<td>70000</td>
</tr>
</tbody>
</table>
Functional Dependencies (FDs)

❖ A functional dependency (FD) has the form: \(X \rightarrow Y\), where \(X\) and \(Y\) are two sets of attributes.
   - Examples: Rating \(\rightarrow\) Hrly_wage, \(AB \rightarrow C\)
   - \(X, Y, Z\) denote attribute sets; \(A, B, C\) single attributes.

❖ FD \(X \rightarrow Y\) is satisfied by relation instance \(r\) if:
   - For all pairs of tuples \(t_1\) and \(t_2 \in r\):
     \((t_1[X] = t_2[X]) \implies (t_1[Y] = t_2[Y])\)
   - i.e., given any two tuples in \(r\), if the \(X\) values agree, then the \(Y\) values must also agree.

❖ The FD \(X \rightarrow Y\) is NOT satisfied by \(r\) if:
   - There exists a pair of tuples \(t_1\) and \(t_2\) in \(r\) such that
     \(t_1[X] = t_2[X]\) but \(t_1[y] \neq t_2[y]\)
   - i.e., we can find two tuples in \(r\), such that \(X\) values agree, but \(Y\) values don’t.
Functional Dependencies (FDs)

❖ The FD holds over relation R if, for every (allowable) instance \( r \) of R, \( r \) satisfies the FD.

❖ An FD, as an integrity constraint, is a statement about all allowable relation instances.

❖ Given some instance \( r1 \) of R, we can check if it violates some FD \( f \) or not
  - But we cannot infer \( f \) by looking at an instance!
  - (Note: in an analytic database, we actually try to infer causality from instances... which makes FD’s and normalization challenging there!)

❖ FD’s determined from application semantics
  - e.g., “Its company policy that rating determines salary”
  - as for all integrity constraints!
Decomposition Example: Hourly_Emps

❖ Two FDs on Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked); denoted: SNLRWH
  - ssn is the key:  S → SNLRWH
  - rating determines hrly_wages:  R → W

❖ Decomposition & its effect on these FDs
  - SNLRWH is replaced by SNLRH and RW
  - R → W is isolated in a separate relation RW
  - SNLRH has FD:  S → SNLRH
## Instance-Level Decomposition

### SNLRWH

<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>

### SNLRH

<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>
Decomposition at the ER Diagram Level

Entity set Hourly_Emps

A decomposed E-R design
**FD Examples**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

How many possible FDs totally on this relation instance? **49.**

Possible FDs with A as the left side:

<table>
<thead>
<tr>
<th></th>
<th>Satisfied by this relation instance?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A→A</td>
<td>Yes</td>
</tr>
<tr>
<td>A→B</td>
<td>Yes</td>
</tr>
<tr>
<td>A→C</td>
<td>No</td>
</tr>
<tr>
<td>A→AB</td>
<td>Yes</td>
</tr>
<tr>
<td>A→AC</td>
<td>No</td>
</tr>
<tr>
<td>A→BC</td>
<td>No</td>
</tr>
<tr>
<td>A→ABC</td>
<td>No</td>
</tr>
</tbody>
</table>

7 non-empty subsets of \(\{A,B,C\}\) on the left x 7 non-empty subsets of \(\{A,B,C\}\) on the right
FD Examples (cont.)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FD</th>
<th>Satisfied by this relation instance?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C→B</td>
<td>Yes</td>
</tr>
<tr>
<td>C→AB</td>
<td>No</td>
</tr>
<tr>
<td>B→C</td>
<td>No</td>
</tr>
<tr>
<td>B→B</td>
<td>Yes</td>
</tr>
<tr>
<td>AC→B</td>
<td>Yes</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Relationship between FDs and Keys

❖ Let \( \mathcal{R} \) be the set of all attributes in relation \( R \), and \( K \) be a key
  - \( X \rightarrow \mathcal{R} \) means \( X \) is a (super)key of \( R \).
  - \( K \rightarrow \mathcal{R} \) is always an FD

❖ For example:
  - Given relation \( R(A, B, C) \).
  - \( A \rightarrow ABC \) means that \( A \) is a key.

❖ Keys have a lot in common with FDs!
  - Big Insight: the LHS (left hand side) of every FD “would like” to be a key in its own relation
Reasoning About FDs

❖ Given some FDs, we can usually infer additional FDs:
   - Employees(ssn, name, lot, did, since)
   - ssn → name, lot, did, since
   - ssn → did, did → lot implies ssn → lot
   - A → BC implies A → B

❖ An FD $f$ is **logically implied by** a set of FDs $F$ if $f$ holds whenever all FDs in $F$ hold.
   - $F^+$ = **closure of $F$** is the set of all FDs that are implied by $F$. 
Reasoning about FDs

- How do we get all the FDs that are logically implied by a given set of FDs?
- 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
Armstrong’s axioms

- Armstrong’s axioms are **sound** and **complete** inference rules for FDs!

  - **Sound**: the set of new FD’s derived using the axioms contains only (no more than) those logically implied by the base set of FD’s
    - i.e., they don’t derive any non-inferable FD’s

  - **Complete**: all FD’s logically implied by the base set of FDs can be derived using the axioms.
    - i.e., they derive all the inferable FD’s
Example of using Armstrong’s Axioms

❖ Couple of additional rules (that follow from AA):
  - **Union**: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)
  - **Decomposition**: If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)

❖ Derive these using Armstrong’s axioms!
  - **Union**:
    - By augmentation: \( XX \rightarrow XY \) and \( XY \rightarrow YZ \)
    - By transitivity: \( XX \rightarrow YZ \), or \( X \rightarrow YZ \)
  - **Decomposition**:
    - By reflexivity: \( YZ \rightarrow Y \)
    - By transitivity: \( X \rightarrow YZ \) and \( YZ \rightarrow Y \) produces \( X \rightarrow Y \)
Reasoning About FDs

- Given Contracts (contractid, supplierid, projectid, deptid, partid, qty, value)
  - C is the key: \( C \rightarrow CSJDPQV \)
  - A project purchases a given part on at most one contract: \( JP \rightarrow C \)
  - Dept purchases at most one part from a supplier: \( SD \rightarrow P \)

- \( \{ JP \rightarrow C, \ C \rightarrow CSJDPQV \} \Rightarrow \{ JP \rightarrow CSJDPQV \} \)
- \( \{ SD \rightarrow P \} \Rightarrow \{ SDJ \rightarrow JP \} \)
- \( \{ SDJ \rightarrow JP, \ JP \rightarrow CSJDPQV \} \Rightarrow \{ SDJ \rightarrow CSJDPQV \} \)

- Three new derived FDs!
Reasoning About FDs

- Can we derive the FD $A \rightarrow E$ from $F = \{A \rightarrow B, \ B \rightarrow C, \ C \ D \rightarrow E \}$?
  - i.e., is $A \rightarrow E$ in $F^+$? (the closure of $F$)

- Computing the closure of a set of FDs can be expensive.
  - Cardinality of closure can be exponential in # attrs!

- Typically, we just want to check if FD $X \rightarrow Y$ is in the closure of a set of FDs $F$.
  - Good news: We don’t have to actually compute $F^+$ to do this!
Testing For Membership in $F^+$

1) Given a set of FD's $F$, and a new FD $X \rightarrow Y$
   Is $X \rightarrow Y$ in $F^+$?

2) To test this, we first compute the attribute closure of $X$ (denoted $X^+$) wrt $F$:
   $X^+ = \text{the set of all attributes } A \text{ such that } X \rightarrow A \text{ is in } F^+$
   There is a linear time algorithm to compute this. (We learn it on the next slide.)

3) Then, check if $Y$ is in $X^+$
   If $Y$ is in $X^+$, then $X \rightarrow Y$ is in $F^+$
Computing $X^+$

- **Inputs:** $F$ (a set of FDs), and an FD: $X \rightarrow Y$
- **Output:** Result=$X^+$
- **Method:**
  - **Step 1:** Result := $X$;
  - **Step 2:** If $R \rightarrow S \in F$, and $R \in$ Result:
    
    Result := $S \cup$ Result
  - **Repeat step 2 until Result cannot be changed,** then output Result.
Example Using $X^+$

- $F=\{A \rightarrow B, AC \rightarrow D, AB \rightarrow C\}$
- Is $A \rightarrow BCD$ an FD in $F^+$?
- Compute: $A^+ = ABCD$
- $ABCD \supseteq BCD$; therefore YES!

Note: we can find all candidate keys in relation $R$ with this algorithm!
  - start with each $X$ in $R$, where $X$ is a single attribute
  - if $X^+ = \text{all attributes in } R$, $X$ is a candidate key for $R$
  - grow $X$ to 2, 3 ... attributes in length, until done (due to minimality)
Computing $F^+$ (Using $X^+$)

Given $F=\{ A \rightarrow B, B \rightarrow C \}$. Compute $F^+$ (with attributes $A, B, C$).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>AB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BC</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ABC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- An entry with ✓ means FD (the row) $\rightarrow$ (the column) is in $F^+$.
- An entry gets ✓ when (the column) is in (the row)$^+$

Attribute closure

<table>
<thead>
<tr>
<th></th>
<th>A$^+$=ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>B$^+$=BC</td>
<td></td>
</tr>
<tr>
<td>C$^+$=C</td>
<td></td>
</tr>
<tr>
<td>AB$^+$=ABC</td>
<td></td>
</tr>
<tr>
<td>AC$^+$=ABC</td>
<td></td>
</tr>
<tr>
<td>BC$^+$=BC</td>
<td></td>
</tr>
<tr>
<td>ABC$^+$=ABC</td>
<td></td>
</tr>
</tbody>
</table>
Check if two sets of FDs are equivalent

Two sets $F$ and $G$ of FDs are equivalent if they logically imply the same set of FDs.
- I.e., if $F^+ = G^+$, then they are equivalent.

For example, $F = \{A \rightarrow B, A \rightarrow C\}$ is equivalent to $G = \{A \rightarrow BC\}$

How to test? Two steps:
- Every FD in $F$ is in $G^+$
- Every FD in $G$ is in $F^+$

These two steps can use the algorithm (many times) for $X^+$
Equivalences of Sets of FDs

❖ Two set of FD’s G and F are equivalent if:

\[ G^+ = F^+ \]

Every FD in G can be inferred from F, and vice versa

❖ Testing Equivalence of FD Sets

1. For each FD \( X \rightarrow Y \) in G, check if \( X \rightarrow Y \in F^+ \) holds
   Compute the closure \( X^+ \) w.r.t. F
   If \( Y \subseteq X^+ \) then \( X \rightarrow Y \in F^+ \)

2. For each FD’s \( W \rightarrow Z \) in F, check if \( W \rightarrow Z \in G^+ \) holds
   Compute the closure \( W^+ \) w.r.t. G
   If \( Z \subseteq W^+ \) then \( W \rightarrow Z \in G^+ \)
Example

Given \( F = \{ \text{BD} \rightarrow \text{E} , \text{C} \rightarrow \text{A} , \text{E} \rightarrow \text{C} , \text{A} \rightarrow \text{E} \} \) and \( G = \{ \text{C} \rightarrow \text{E} , \text{BD} \rightarrow \text{C} \} \). Are \( G \) and \( F \) equivalent?

1. For each FD \( X \rightarrow Y \) in \( G \), does \( X \rightarrow Y \in F^+ \) hold?
   - \( C_F^+ = \{ \text{A,C,E} \} \): \( \text{C} \rightarrow \text{E} \in F^+ \)
   - \( (\text{BD})_F^+ = \{ \text{B,D,E,C,A} \} \): \( \text{BD} \rightarrow \text{C} \in F^+ \)

2. For each FD \( W \rightarrow Z \) in \( F \), does \( W \rightarrow Z \in G^+ \) hold?
   - \( (\text{BD})_G^+ = \{ \text{B,D,C,E} \} \): \( \text{BD} \rightarrow \text{E} \in G^+ \)
   - \( (\text{C})_G^+ = \{ \text{C,E} \} \): \( \text{C} \rightarrow \text{A} \notin G^+ \)

\* F and G are not equivalent!
Summary

❖ We need to understand functional dependencies, and how they can lead to schema design issues:
   - update/insert/delete anomalies due to redundancy
   - intuitively awkward designs

❖ Next week:

❖ **Schema refinement** can alleviate such problems
   - Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).
   - Goal: decompose into a normal form
     ♦ several are possible: e.g., 3NF, BCNF...

❖ Once again: Decomposition should be used judiciously as problems can result:
   - Violate desirable properties of decompositions (lossless-join and dependency-preservation.)
   - Over-normalization
Refining an ER Diagram

- 1st diagram translated:
  Employees(S,N,L,D,S)
  Departments(D,M,B)
  - Lots associated with workers.

- Suppose all workers in a dept are assigned the same lot: $D \rightarrow L$

- Can adjust this to:
  Employees2(S,N,D,S)
  Departments2(D,M,B,L)