Schema Refinement and Normal Forms
Database Design Steps

1. Real world to
2. E/R model to
3. Relational schema to
4. Better relational schema (new!) to
5. Relational DBMS

Step 3 to Step 4 is based on a design theory for relations and is called “normalization”; key goal: eliminate redundancy
## MovieBad(title, year, length, starName)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Bjork</td>
</tr>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Catherine Deneuve</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

**Key?**
### MovieBad(title, year, length, starName)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Bjork</td>
</tr>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Catherine Deneuve</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

**Key?** \{title, year, starName\}

**Problems with this schema?**
### MovieBad(title, year, length, starName)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Bjork</td>
</tr>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Catherine Deneuve</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

**Key?** \{title, year, starName\}

**Problems with this schema?**

- title, year repeated for a movie? Not a problem: both attributes needed to identify movie
MovieBad(title, year, length, starName)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Bjork</td>
</tr>
<tr>
<td>“Dancer in the dark”</td>
<td>2000</td>
<td>140’</td>
<td>Catherine Deneuve</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

Key? {title, year, starName}

Problems with this schema?

- title, year repeated for a movie? Not a problem: both attributes needed to identify a movie
- length repeated for a movie? Yes! title and year determine length, yet current schema allows for different lengths for the same movie (redundancy!), which is a problem
- Also, what if a movie doesn't have any stars? Where do we store its length?
3 Types of Anomalies in MovieBad Schema

MovieBad(title, year, length, starName)

- **Redundancy**: length for a movie stored multiple times
- **Update anomaly**: inconsistencies possible if we update length of a movie
- **Deletion anomaly**: if we delete a star for a movie, we might accidentally lose the movie length as well (and all records of the movie)
Better Design to Replace MovieBad(title, year, length, starName)?
Better Design to Replace MovieBad(title, year, length, starName)

- Movie(title, year, length)
- MovieStar(title, year, starName)

Good design with:

- no anomalies and
- ability to “reconstruct” exactly all original information (how?)
Better Design to Replace MovieBad(title, year, length, starName)

• Movie(title, year, length)
• MovieStar(title, year, starName)

Good design with:

• no anomalies and
• ability to “reconstruct” exactly all original information (how?) Natural join of both relations!
Functional Dependencies (FDs)

A functional dependency is a special kind of constraint for a relation:

- is based on knowledge of real world
- must always hold for the relation

Example over Movie(title, year, length) table:

- Functional dependency \textit{title, year} \textit{→} \textit{length} holds
- Meaning: any two Movie tuples that agree on both title and year (i.e., on left side of FD) must also agree on length (i.e., on right side of FD)
A functional dependency $F$ for a relation $R$ consists of one or more attributes $A_1$, $A_2$, ..., $A_m$ of $R$, followed by symbol $\rightarrow$, followed by one or more attributes $B_1$, $B_2$, ..., $B_n$ of $R$:

$A_1, A_2, ..., A_m \rightarrow B_1, B_2, ..., B_n$
Functional Dependency: Definition

• A **functional dependency** $F$ for a relation $R$ consists of one or more attributes $A_1, A_2, \ldots, A_m$ of $R$, followed by symbol $\rightarrow$, followed by one or more attributes $B_1, B_2, \ldots, B_n$ of $R$:

$$A_1, A_2, \ldots, A_m \rightarrow B_1, B_2, \ldots, B_n$$

• We say that $F$ holds for $R$ if, for every legal instance of $R$, whenever tuples $t$ and $u$ of $R$ agree on all the values of $A_1, A_2, \ldots, A_m$, then $t$ and $u$ must also agree on all the values of $B_1, B_2, \ldots, B_n$

• For brevity, we often omit the commas that separate attributes (so we write, say, $A_1 A_2 \ldots A_m \rightarrow B_1 B_2 \ldots B_n$)
Some Example FDs for MovieBad
MovieBad(title, year, length, starName)
Some Example FDs for MovieBad

MovieBad(title, year, length, starName)

• title, year → length
• title → title
• title, year, starName → length
• title, year, starName → length, year
• ...

Some of these are trivial (e.g., title → title); the rest are real-world constraints for the data (e.g., title, year → length)
Reasoning About FDs

Consider relation EmpsBad(ssn, name, did, rank, sal) with 2 given FDs:

- ssn → name, did, rank
- did, rank → sal

(We’ll see later why this schema is “bad”)

Any other FDs that hold, given the 2 FDs above?
Reasoning About FDs

Consider relation \textit{EmpsBad}(ssn, name, did, rank, sal) with 2 given FDs:

- \( ssn \rightarrow name, did, rank \)
- \( did, rank \rightarrow sal \)

(We’ll see later why this schema is “bad”)

Any other FDs that hold, given the 2 FDs above?

We can conclude that \textbf{ssn} \( \rightarrow \) \textbf{sal} also holds (plus several others)
“Closure” of a Set of FDs

- Consider a set of FDs $S$ over a given relation.
- A FD $F$ is “implied” by $S$ if $F$ holds whenever all FDs in $S$ hold.
- The closure of $S$, denoted $S^+$, is the set of all FDs implied by $S$. 
“Closure” of a Set of FDs

- Consider a set of FDs $S$ over a given relation
- A FD $F$ is “implied” by $S$ if $F$ holds whenever all FDs in $S$ hold
- The closure of $S$, denoted $S^+$, is the set of all FDs implied by $S$

Given a relation and a set of FDs $S$, calculating $S^+$ is important to decide if the relation schema is in “good form” or not
One Way to Reason about FDs

- To compute the closure of a set of FDs, we can use Armstrong’s axioms:
One Way to Reason about FDs

• To compute the closure of a set of FDs, we can use Armstrong’s axioms:
  
  • **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$
  
  • **Augmentation**: If $X \rightarrow Y$ holds, then $X \cup Z \rightarrow Y \cup Z$ also holds for any attribute set $Z$
  
  • **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ hold, then so does $X \rightarrow Z$

• Armstrong’s axioms are a **sound** and **complete** set of inference rules for FDs
Using Armstrong’s Axioms: An Example

R(A, B, C, D, E) is a relation with given set S of FDs:
(1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold? (In other words, is AD → E in S⁺?)
Using Armstrong’s Axioms: An Example

R(A, B, C, D, E) is a relation with given set S of FDs: (1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold? (In other words, is AD → E in S⁺?)

Answer: Yes! Use Armstrong’s axioms to prove it:
Using Armstrong’s Axioms: An Example

R(A, B, C, D, E) is a relation with given set S of FDs: (1) A → B, (2) B → C, and (3) CD → E

Question: Does $AD \rightarrow E$ hold for R given that (1), (2), and (3) hold? (In other words, is $AD \rightarrow E$ in $S^+$?)

Answer: Yes! Use Armstrong’s axioms to prove it:
- From (1) and (2), and transitivity: (4) $A \rightarrow C$
Using Armstrong’s Axioms: An Example

R(A, B, C, D, E) is a relation with given set S of FDs: 
(1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold? (In other words, is AD → E in S⁺?)

Answer: Yes! Use Armstrong’s axioms to prove it:
• From (1) and (2), and transitivity: (4) A → C
• From (4) and augmentation: (5) AD → CD
Using Armstrong’s Axioms: An Example

R(A, B, C, D, E) is a relation with given set S of FDs:
(1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold? (In other words, is AD → E in S⁺?)

Answer: Yes! Use Armstrong’s axioms to prove it:

• From (1) and (2), and transitivity: (4) A → C
• From (4) and augmentation: (5) AD → CD
• From (5) and (3), and transitivity: AD → E
“Closure” of Set of Attributes

• As an alternative to the axioms for reasoning about what FDs hold, we can use the “closure” of a set of attributes

• Consider a relation R, a set of FDs S for R, and a set of attributes \{A_1, \ldots, A_k\} of R

• Then the closure of \{A_1, \ldots, A_k\}, denoted \{A_1, \ldots, A_k\}^+, consists of all attributes B in R such that \(A_1, A_2, \ldots, A_k \rightarrow B\)
Closure of Set of Attributes: An Algorithm

To compute the closure of \{A_1, \ldots, A_k\}, \{A_1, \ldots, A_k\}^+, for a relation R with respect to a set of FDs S for R:
Closure of Set of Attributes: An Algorithm

To compute the closure of \{A_1, \ldots, A_k\}, \{A_1, \ldots, A_k\}^+, for a relation R with respect to a set of FDs S for R:

- Start with \{A_1, \ldots, A_k\}
- Repeat until no change:
  - If current set of attributes includes the full left side of a FD in S, then add all the right side attributes of the FD to the set
- Return the set of attributes
Using Closure of Set of Attributes: Example Revisited

R(A, B, C, D, E) is a relation with given FDs:
(1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold?
Using Closure of Set of Attributes: Example Revisited

R(A, B, C, D, E) is a relation with given FDs:
(1) A → B, (2) B → C, and (3) CD → E
Question: Does \textbf{AD} → E hold for R given that (1), (2), and (3) hold?
Answer: Compute \{A, D\}⁺ and check if E is in the closure
Using Closure of Set of Attributes: Example Revisited

R(A, B, C, D, E) is a relation with given FDs:
(1) A → B, (2) B → C, and (3) CD → E
Question: Does $AD \rightarrow E$ hold for R given that (1), (2), and (3) hold?
Answer: Compute $\{A, D\}^+$ and check if E is in the closure
• Start with closure=$\{A, D\}$
R(A, B, C, D, E) is a relation with given FDs:
(1) \( A \rightarrow B \), (2) \( B \rightarrow C \), and (3) \( CD \rightarrow E \)

Question: Does \( AD \rightarrow E \) hold for \( R \) given that (1), (2), and (3) hold?

Answer: Compute \( \{A, D\}^+ \) and check if E is in the closure

- Start with closure=\( \{A, D\} \)
- closure includes left side of (1) (i.e., attribute A), so add right side (i.e., attribute B) to closure: closure=\( \{A, B, D\} \)
Using Closure of Set of Attributes: Example Revisited

R(A, B, C, D, E) is a relation with given FDs:
(1) A → B, (2) B → C, and (3) CD → E
Question: Does \( AD \) → E hold for R given that (1), (2), and (3) hold?

Answer: Compute \( \{A, D\}^+ \) and check if E is in the closure

- Start with closure=\{A, D\}
- closure includes left side of (1) (i.e., attribute A), so add right side (i.e., attribute B) to closure: closure=\{A, B, D\}
- closure includes left side of (2) (i.e., attribute B), so add C to closure: closure=\{A, B, C, D\}
- closure includes left side of (3) (i.e., attributes C and D), so add E to closure: closure=\{A, B, C, D, E\}
- No more changes possible, so return \( \{A, D\}^+ = \{A, B, C, D, E\} \)
Using Closure of Set of Attributes: Example Revisited

R(A, B, C, D, E) is a relation with given FDs:
(1) A → B, (2) B → C, and (3) CD → E

Question: Does AD → E hold for R given that (1), (2), and (3) hold?

Answer: Compute \{A, D\}^+ and check if E is in the closure

- Start with closure={A, D}
- closure includes left side of (1) (i.e., attribute A), so add right side (i.e., attribute B) to closure: closure={A, B, D}
- closure includes left side of (2) (i.e., attribute B), so add C to closure: closure={A, B, C, D}
- closure includes left side of (3) (i.e., attributes C and D), so add E to closure: closure={A, B, C, D, E}
- No more changes possible, so return \{A, D\}^+={A, B, C, D, E}
- Because E is in \{A, D\}^+, if follows that AD → E holds
Revisiting Definition of “Key” Using FDs

Consider a relation $R(A_1, A_2, \ldots, A_n)$ and a set of attributes $K$ of $R$ (i.e., $K \subseteq \{A_1, A_2, \ldots, A_n\}$)

**K is a key for R** if and only if:

1. $K \rightarrow A_1, A_2, \ldots, A_n$ (i.e., $K$ “functionally determines” all attributes of $R$)
2. No strict subset of $K$ satisfies Condition 1 (i.e., $K$ is minimal)
Revisiting Definition of “Key” Using FDs

Consider a relation $R(A_1, A_2, \ldots, A_n)$ and a set of attributes $K$ of $R$ (i.e., $K \subseteq \{A_1, A_2, \ldots, A_n\}$)

$K$ is a key for $R$ if and only if:

1. $K \rightarrow A_1, A_2, \ldots, A_n$ (i.e, $K$ “functionally determines” all attributes of $R$)
2. No strict subset of $K$ satisfies Condition 1 (i.e., $K$ is minimal)

In general, if a set of attributes $K'$ satisfies Condition 1 then $K'$ is a superkey for $R$

If additionally $K'$ has the minimality property, then $K'$ is also a key (so every key is a superkey but not the other way around)
Identifying All Keys for a Relation

Consider a relation $R(A_1, \ldots, A_n)$ and a given set of functional dependencies $S$ that hold over $R$. **How do we determine all keys for $R$?**
Identifying All Keys for a Relation

Consider a relation $R(A_1, \ldots, A_n)$ and a given set of functional dependencies $S$ that hold over $R$. **How do we determine all keys for $R$?**

**Task:** Identify every set $K$ of attributes of $R$ such that:

1. $K \rightarrow$ all attributes of $R$
2. $K$ is minimal with such property (i.e., no strict subset of $K$ functionally determines all attributes of $R$)
Identifying All Keys for a Relation

Consider a relation $R(A_1, \ldots, A_n)$ and a given set of functional dependencies $S$ that hold over $R$. How do we determine all keys for $R$?

Task: Identify every set $K$ of attributes of $R$ such that:

1. $K \rightarrow$ all attributes of $R$
2. $K$ is minimal with such property (i.e., no strict subset of $K$ functionally determines all attributes of $R$)

Algorithm for Finding All Keys for Relation $R$
(note it is exponential in the number of attributes $n$ of the relation)

For $i = 1$ to $n$:

- Consider each set of attributes $C_i$ with exactly $i$ attributes of $R$
- If $C_i$ includes a (previously found) key, then ignore $C_i$ ($C_i$ could not satisfy minimality)
- Otherwise, compute the closure of $C_i$: if $C_i^+ = \{A_1, \ldots, A_n\}$, then $C_i$ is a key for $R$
Is a Relation Schema “Good”?

• “Normal forms” determine which relation schemas are “good,” in the sense of avoiding certain kinds of redundancy

• **Boyce-Codd Normal Form (BCNF):** Consider a relation R and a set of functional dependencies S for R

R is in BCNF with respect to FD set S if
Is a Relation Schema “Good”?

- “Normal forms” determine which relation schemas are “good,” in the sense of avoiding certain kinds of redundancy

- **Boyce-Codd Normal Form (BCNF):** Consider a relation R and a set of functional dependencies S for R

  R is in BCNF with respect to FD set S if for every FD \( X \rightarrow A \) that holds over R:
  - \( X \rightarrow A \) is **trivial**, meaning that A is included in X, or
  - \( X \) contains a key (i.e., **X is a superkey**)

44
Is EmpsBad in BCNF?

EmpsBad(ssn, name, did, rank, sal)

Given FDs:

- ssn → name, did, rank
- did, rank → sal
Is EmpsBad in BCNF?

EmpsBad(ssn, name, did, rank, sal)

Given FDs:

- $ssn \rightarrow name, did, rank$
- $did, rank \rightarrow sal$

First, we can use the algorithm above to identify all keys of the relation: $\{ssn\}$ is the only key

$\{ssn\}^+$ includes all attributes of the relation and, of course, $\{ssn\}$ is minimal; furthermore, we should show that no other attribute set satisfies these two conditions
Is EmpsBad in BCNF?

EmpsBad(ssn, name, did, rank, sal)

Given FDs:

- ssn → name, did, rank
- did, rank → sal

First, we can use the algorithm above to identify all keys of the relation: \{ssn\} is the only key

\{ssn\}^+ includes all attributes of the relation and, of course, \{ssn\} is minimal; furthermore, we should show that no other attribute set satisfies these two conditions

Then EmpsBad not in BCNF: did, rank → sal holds, it is nontrivial, and its left side is not a superkey
Is EmpsBad in BCNF?

EmpsBad(ssn, name, did, rank, sal)

Given FDs:

- ssn → name, did, rank
- did, rank → sal

First, we can use the algorithm above to identify all keys of the relation: \{ssn\} is the only key

\{ssn\}^+ includes all attributes of the relation and, of course, \{ssn\} is minimal; furthermore, we should show that no other attribute set satisfies these two conditions

Then EmpsBad not in BCNF: did, rank → sal holds, it is nontrivial, and its left side is not a superkey

Why is this a problem? sal for each <did, rank> pair is potentially stored many times in relation ...
Another Example

MovieBad(title, year, length, starName)

Given FD: title, year → length
Another Example

MovieBad(title, year, length, starName)

Given FD: title, year $\rightarrow$ length

- Only key of the relation is \{title, year, starName\}
- **title, year $\rightarrow$ length** holds for MovieBad, it is nontrivial, and its left side doesn’t include a key (it’s missing starName)
- Hence MovieBad is not in BCNF
Handling BCNF Violations

As we will see, when a relation is not in Boyce-Codd Normal Form, we can decompose it into “smaller” relations to avoid redundancy:

• Split `EmpsBad(ssn, name, did, rank, sal)` into:
  
  (“Offending” FD was: `did, rank → sal`)

• Split `MovieBad(title, year, length, starName)` into:
  
  (“Offending” FD was: `title, year → length`)

Handling BCNF Violations

As we will see, when a relation is not in Boyce-Codd Normal Form, we can decompose it into “smaller” relations to avoid redundancy:

• Split EmpsBad(ssn, name, did, rank, sal) into:
  • Employee(ssn, name, did, rank) and
  • Salary(did, rank, sal)
  ("Offending" FD was: did, rank → sal)

• Split MovieBad(title, year, length, starName) into:
  • Movie(title, year, length)
  • MovieStar(title, year, starName)
  ("Offending" FD was: title, year → length)
Algorithm for Decomposing a Relation into BCNF Relations

- **Input**: relation schema $R$ and set $S$ of FDs for $R$
- **Output**: a decomposition of $R$ into BCNF relations
Algorithm for Decomposing a Relation into BCNF Relations

- **Input**: relation schema R and set S of FDs for R
- **Output**: a decomposition of R into BCNF relations

1. Identify all keys for R based on set S of FDs for R
   // use attribute closure algorithm; NP-complete problem
2. Repeat until no more BCNF violations:
   - Pick any R'(Y) with a (nontrivial) FD X → A that violates BCNF, where A is an individual attribute
   - Decompose R' into R<sub>1</sub>(X ∪ {A}) and R<sub>2</sub>(Y − {A})
   - Compute FDs that hold over R<sub>1</sub> and R<sub>2</sub> // use Armstrong’s axioms or attribute closure algorithm (preferred) for every subset of attributes of R<sub>1</sub> and R<sub>2</sub> over the R’ FDs
   - Identify all keys for R<sub>1</sub> and R<sub>2</sub> based on their FDs

Final decomposed relations may be different depending on which violating FD is chosen in each iteration of Step 2, but all decompositions will be in BCNF
Does BCNF Decomposition Algorithm Guarantee a Good Decomposition?

• Removes anomalies? Yes
• Enables reconstruction of original relation? Yes, through natural join
  • Can we get too many tuples in natural join result? No
  • Can we get too few tuples in natural join result? No

We will see why, but we need a few definitions first…
Valid Relation Decomposition

Let \( R(A_1, A_2, \ldots, A_n) \) be a relation. Then:

- \( R_1 = \pi_U(R) \) and
- \( R_2 = \pi_V(R) \)

form a decomposition of \( R \) if \( U \cup V = \{A_1, A_2, \ldots, A_n\} \)

That is, \( R_1 \) and \( R_2 \) together have all attributes of \( R \), nothing more, nothing less
Example of Valid Relation Decomposition

MovieBad(title, year, length, starName)
• Movie(title, year, length) and
• MovieStar(title, year, starName)

form a valid decomposition of MovieBad: Movie and MovieStar include all attributes of MovieBad, and nothing else
Valid Relation Decomposition

How about:

- Movie(title, year, length) and
- MovieStar2(title, starName)?

Also a valid decomposition, but this is a **bad** one. Why?
### MovieBad

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

\[
\text{Movie} = \Pi_{\text{title, year, length}}(\text{MovieBad})
\]

\[
\text{MovieStar2} = \Pi_{\text{title, starName}}(\text{MovieBad})
\]

\[
\text{Movie} \Join \text{MovieStar2}?
\]
MovieBad

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

Movie = \( \Pi_{\text{title, year, length}} \) (MovieBad)

MovieStar2 = \( \Pi_{\text{title, starName}} \) (MovieBad)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
</tbody>
</table>

Movie \( \Join \) MovieStar2: 2 spurious tuples that shouldn’t be there!

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Donat</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Powell</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1935</td>
<td>86’</td>
<td>Robert Powell</td>
</tr>
<tr>
<td>“The 39 steps”</td>
<td>1978</td>
<td>92’</td>
<td>Robert Donat</td>
</tr>
</tbody>
</table>
Valid Relation Decomposition

How about:
• Movie(title, year, length) and
• MovieStar2(title, starName)?

Also a valid decomposition, but this is a bad one.
Why? We cannot accurately “reconstruct” MovieBad from Movie and MovieStar2 (natural join might contain extra, “spurious” tuples not in MovieBad …)

Source of problem: Intuitively, Movie and MovieStar2 don’t share “enough attributes” (“title” alone not sufficient)
Lossless Join Decompositions

A decomposition of a relation $R$ into:

- $R_1 = \pi_U(R)$ and
- $R_2 = \pi_V(R)$

is **lossless join** with respect to a given set of FDs $S$ if, for every instance of $R$ that satisfies $S$, $R = R_1 \Join R_2$

Note: It is always the case that $R \subseteq R_1 \Join R_2$ for a valid, not necessarily lossless join decomposition of $R$ into $R_1$ and $R_2$

When is a decomposition guaranteed to be lossless join?
Lossless Join Decompositions: A Theorem

**Theorem:** The decomposition of a relation $R$ into:
- $R_1 = \pi_U(R)$ and
- $R_2 = \pi_V(R)$
is lossless join with respect to a given set of FDs $S$ if and only if $S^+$, the closure of $S$, contains either:
- $U \cap V \rightarrow U$ or
- $U \cap V \rightarrow V$

The BCNF decomposition algorithm always produces lossless join decompositions (good exercise: prove why)
Decompositions and Efficiency

• So far, we know that we want lossless join decompositions

• But how about efficiency? To check/enforce FDs over a decomposed relation, we sometimes need to take the natural join of the smaller parts, which has a cost

• Hence we want to figure out which decompositions, called “dependency preserving,” avoid such natural joins to check/enforce FDs
BCNF and Dependency Preservation

• We can always decompose a relation into BCNF relations with a lossless join decomposition, using the algorithm that we have seen

• However, sometimes there is no dependency preserving decomposition into BCNF

• This motivates the introduction of another normal form, 3NF, that allows for a bit more redundancy than BCNF does

• We can always decompose a relation into 3NF relations with a lossless join and dependency preserving decomposition

• We will not cover 3NF in our class
Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good goal.
- If a relation is not in BCNF, we can always decompose it into a collection of BCNF relations (algorithm covered in class), such that the decomposition:
  - is guaranteed to be lossless join, but
  - might not be dependency preserving (i.e., we might need to perform natural joins to check that FDs are satisfied)
- 3NF tolerates a bit more redundancy than BCNF
- If a relation is not in 3NF, we can always decompose it into a collection of 3NF relations such that the decomposition is guaranteed to be both lossless join and dependency preserving (3NF not covered in our course)