CMSC424: Database Design Normalization

February 26, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu
Desiderata

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No “inability to represent information”
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible. We sometimes relax these for:

*simpler schemas, and fewer joins during queries.*
## FDs: Example 1

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>StarName</th>
<th>Birthdate</th>
<th>producerC#</th>
<th>Producer address</th>
<th>Prdocuer name</th>
<th>netWorth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane Crazy</td>
<td>1927</td>
<td>6</td>
<td>NULL</td>
<td>NULL</td>
<td>WD100</td>
<td>Mickey Rd</td>
<td>Walt Disney</td>
<td>100000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>H. Ford</td>
<td>7/13/42</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>M. Hamill</td>
<td>9/25/51</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>C. Fisher</td>
<td>10/21/56</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>F. Wray</td>
<td>9/15/07</td>
<td>MC100</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>N. Watts</td>
<td>9/28/68</td>
<td>PJ100</td>
<td>Middle Earth</td>
<td>Peter Jackson</td>
<td>10^8</td>
</tr>
<tr>
<td>State Name</td>
<td>State Code</td>
<td>State Population</td>
<td>County Name</td>
<td>County Population</td>
<td>Senator Name</td>
<td>Senator Elected</td>
<td>Senator Born</td>
<td>Senator Affiliation</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>------------------</td>
<td>-------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Autauga</td>
<td>54571</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Baldwin</td>
<td>182265</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Barbour</td>
<td>27457</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Autauga</td>
<td>54571</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Baldwin</td>
<td>182265</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Barbour</td>
<td>27457</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
</tbody>
</table>
### FDs: Example 3

<table>
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
<th>Dept Name</th>
<th>Credits</th>
<th>Section ID</th>
<th>Semester</th>
<th>Year</th>
<th>Building</th>
<th>Room No.</th>
<th>Capacity</th>
<th>Time Slot ID</th>
</tr>
</thead>
</table>

**Functional dependencies**

- `course_id → title, dept_name, credits`
- `building, room_number → capacity`
- `course_id, section_id, semester, year → building, room_number, time_slot_id`
Functional Dependencies

- Let $R$ be a relation schema and
  \[ \alpha \subseteq R \text{ and } \beta \subseteq R \]
- The functional dependency
  \[ \alpha \rightarrow \beta \]
  holds on $R$ iff for any legal relations $r(R)$, whenever two tuples $t_1$ and $t_2$ of $r$ have same values for $\alpha$, they have same values for $\beta$.
  \[ t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta] \]
- Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</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, $A \rightarrow B$ does NOT hold, but $B \rightarrow A$ does hold.
Functional Dependencies

Difference between holding on an *instance* and holding on *all legal relation*

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

*Title → Year* holds on this instance

*Is this a true functional dependency? No.*

*Two movies in different years can have the same name.*

Can’t draw conclusions based on a *single instance*

Need to use domain knowledge to decide which FDs hold
FDs and Redundancy

- Consider a table: R(A, B, C):
  - With FDs: B → C, and A → BC
  - So “A” is a Key, but “B” is not
- So: there is a FD whose left hand side is not a key
  - Leads to redundancy

Since B is not unique, it may be duplicated
Every time B is duplicated, so is C

Not a problem with A → BC
A can never be duplicated

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a3</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a4</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>a5</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>a6</td>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>a7</td>
<td>b4</td>
<td>c1</td>
</tr>
</tbody>
</table>

Not a duplication → Two different tuples just happen to have the same value for C
FDs and Redundancy

Better to split it up

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
</tr>
<tr>
<td>a3</td>
<td>b1</td>
</tr>
<tr>
<td>a4</td>
<td>b2</td>
</tr>
<tr>
<td>a5</td>
<td>b2</td>
</tr>
<tr>
<td>a6</td>
<td>b3</td>
</tr>
<tr>
<td>a7</td>
<td>b4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>b4</td>
<td>c1</td>
</tr>
</tbody>
</table>

Not a duplication → Two different tuples just happen to have the same value for C
A relation schema \( R \) is “in BCNF” if:
- Every functional dependency \( A \rightarrow B \) that holds on it is **EITHER**:
  1. Trivial **OR**
  2. \( A \) is a superkey of \( R \)

**Why is BCNF good?**
- Guarantees that there can be no redundancy because of a functional dependency
- Consider a relation \( r(A, B, C, D) \) with functional dependency \( A \rightarrow B \) and two tuples: \((a1, b1, c1, d1)\), and \((a1, b1, c2, d2)\)
  - \( b1 \) is repeated because of the functional dependency
  - BUT this relation is not in BCNF
  - \( A \rightarrow B \) is neither trivial nor is \( A \) a superkey for the relation
Functional Dependencies

- Functional dependencies and keys
  - A key constraint is a specific form of a FD.
  - E.g. if $A$ is a superkey for $R$, then: $A \rightarrow R$
  - Similarly for candidate keys and primary keys.

- Deriving FDs
  - A set of FDs may imply other FDs
  - e.g. If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$
  - We will see a formal method for inferring this later
1. A relation instance \( r \) satisfies a set of functional dependencies, \( F \), if the FDs hold on the relation

2. \( F \) holds on a relation schema \( R \) if no legal (allowable) relation instance of \( R \) violates it

3. A functional dependency, \( A \rightarrow B \), is called trivial if:
   - \( B \) is a subset of \( A \)
   - e.g. Movieyear, length \( \rightarrow \) length

4. Given a set of functional dependencies, \( F \), its closure, \( F^+ \), is all the FDs that are implied by FDs in \( F \).
Approach

1. We will encode and list all our knowledge about the schema
   ◦ Functional dependencies (FDs)
   ◦ Also:
     • Multi-valued dependencies (briefly discuss later)
     • Join dependencies etc...

2. We will define a set of rules that the schema must follow to be considered good
   ◦ “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, ...
   ◦ A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema
A relation schema $R$ is “in BCNF” if:

- Every functional dependency $A \rightarrow B$ that holds on it is either:
  1. Trivial OR
  2. $A$ is a superkey of $R$

Why is BCNF good?

- Guarantees that there can be no redundancy because of a functional dependency
- Consider a relation $r(A, B, C, D)$ with functional dependency $A \rightarrow B$ and two tuples: $(a1, b1, c1, d1)$, and $(a1, b1, c2, d2)$
  - $b1$ is repeated because of the functional dependency
  - BUT this relation is not in BCNF
  - $A \rightarrow B$ is neither trivial nor is $A$ a superkey for the relation
Why does redundancy arise?

1. Given a FD, $A \rightarrow B$, if $A$ is repeated ($B - A$) has to be repeated
2. If rule 1 is satisfied, ($B - A$) is empty, so not a problem.
3. If rule 2 is satisfied, then $A$ can’t be repeated, so this doesn’t happen either

Hence no redundancy because of FDs

- Redundancy may exist because of other types of dependencies
  - Higher normal forms used for that (specifically, 4NF)
  - Data may naturally have duplicated/redundant data
    - We can’t control that unless a FD or some other dependency is defined
Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs); Multi-valued dependencies; Join dependencies etc…

2. We will define a set of rules that the schema must follow to be considered good
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema
   - Through lossless decomposition (splitting)
   - Or direct construction using the dependencies information
What if the schema is not in BCNF?

- Decompose (split) the schema into two pieces.

From the previous example: split the schema into:

- \( r_1(A, B), \ r_2(A, C, D) \)
  - The first schema is in BCNF, the second one may not be (and may require further decomposition)
  - No repetition now: \( r_1 \) contains \((a_1, b_1)\), but \( b_1 \) will not be repeated

Careful: you want the decomposition to be lossless

- \( No \ information \ should \ be \ lost \)
  - The above decomposition is lossless
  - We will define this more formally later
Mechanisms and definitions to work with FDs

- Closures, candidate keys, canonical covers etc...
- Armstrong axioms

Decompositions

- Loss-less decompositions, Dependency-preserving decompositions

BCNF

- How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem
1. Closure

- Given a set of functional dependencies, $F$, its closure, $F^+$, is all FDs that are implied by FDs in $F$.
  - *e.g.* If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$

- We can find $F^+$ by applying Armstrong’s Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$  \hspace{1cm} (reflexivity)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$  \hspace{1cm} (augmentation)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$  \hspace{1cm} (transitivity)

- These rules are
  - sound (generate only functional dependencies that actually hold)
  - complete (generate all functional dependencies that hold)
Additional rules

- If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta \gamma$ (union)
- If $\alpha \rightarrow \beta \gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ (decomposition)
- If $\alpha \rightarrow \beta$ and $\gamma \beta \rightarrow \delta$, then $\alpha \gamma \rightarrow \delta$ (pseudotransitivity)

- The above rules can be inferred from Armstrong’s axioms.
Example

  
  $F = \{ \begin{align*} 
  &A \to B \\
  &A \to C \\
  &CG \to H \\
  &CG \to I \\
  &B \to H \end{align*} \}$

- Some members of $F^+$
  - $A \to H$
    - by transitivity from $A \to B$ and $B \to H$
  - $AG \to I$
    - by augmenting $A \to C$ with $G$, to get $AG \to CG$
      and then transitivity with $CG \to I$
  - $CG \to HI$
    - by augmenting $CG \to I$ to infer $CG \to CGI$, and augmenting of $CG \to H$ to infer $CGI \to HI$, and then transitivity
2. Closure of an attribute set

- Given a set of attributes $A$ and a set of FDs $F$, \textit{closure of $A$ under $F$} is the set of all attributes implied by $A$

- In other words, the largest $B$ such that: $A \rightarrow B$

- Redefining \textit{super keys}:
  - The closure of a super key is the entire relation schema

- Redefining \textit{candidate keys}:
  1. It is a super key
  2. No subset of it is a super key
Computing the closure for $A$

- Simple algorithm

1. Start with $B = A$.
2. Go over all functional dependencies, $\beta \rightarrow \gamma$, in $F^+$
3. If $\beta \subseteq B$, then
   - Add $\gamma$ to $B$
4. Repeat till $B$ changes
Example

- \( R = (A, B, C, G, H, I) \)
- \( F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \} \)

- \((AG)^+?\)
  - 1. result = AG
  - 2. result = ABCG \((A \rightarrow C \text{ and } A \rightarrow B)\)
  - 3. result = ABCGH \((CG \rightarrow H \text{ and } CG \subseteq AGBC)\)
  - 4. result = ABCGHI \((CG \rightarrow I \text{ and } CG \subseteq AGBCH)\)

- Is \((AG)\) a candidate key?
  1. It is a super key.
  2. \((A^+) = ABCH, (G^+) = G.\)

**YES.**
Uses of attribute set closures

- Determining superkeys and candidate keys
- Determining if $A \rightarrow B$ is a valid FD
  - Check if $A+$ contains $B$
- Can be used to compute $F+$
3. Extraneous Attributes

- Consider \( F \), and a functional dependency, \( A \rightarrow B \).

- “Extraneous”: Are there any attributes in \( A \) or \( B \) that can be safely removed?
  
  *Without changing the constraints implied by \( F \)*

- Example: Given \( F = \{ A \rightarrow C, AB \rightarrow CD \} \)
  
  - \( C \) is extraneous in \( AB \rightarrow CD \) since \( AB \rightarrow C \) can be inferred even after deleting \( C \)
  
  - i.e., given: \( A \rightarrow C \), and \( AB \rightarrow D \), we can use Armstrong Axioms to infer \( AB \rightarrow CD \)
4. Canonical Cover

- A *canonical cover* for $F$ is a set of dependencies $F_c$ such that
  - $F$ logically implies all dependencies in $F_c$, and
  - $F_c$ logically implies all dependencies in $F$, and
  - No functional dependency in $F_c$ contains an extraneous attribute, and
  - Each left side of functional dependency in $F_c$ is unique

- In some (vague) sense, it is a *minimal* version of $F$

- Read up algorithms to compute $F_c$
Mechanisms and definitions to work with FDs
  ◦ Closures, candidate keys, canonical covers etc...
  ◦ Armstrong axioms

Decompositions
  ◦ Loss-less decompositions, Dependency-preserving decompositions

BCNF
  ◦ How to achieve a BCNF schema

BCNF may not preserve dependencies

3NF: Solves the above problem

BCNF allows for redundancy

4NF: Solves the above problem
**Loss-less Decompositions**

- Definition: A decomposition of $R$ into $(R1, R2)$ is called *lossless* if, for all legal instance of $r(R)$:
  
  $$ r = \Pi_{R1}(r) \Pi_{R2}(r) $$

- In other words, projecting on $R1$ and $R2$, and joining back, results in the relation you started with.

- Rule: A decomposition of $R$ into $(R1, R2)$ is *lossless*, iff:
  
  $$ R1 \cap R2 \rightarrow R1 \quad \text{or} \quad R1 \cap R2 \rightarrow R2 $$

  in $F+$. 

Is it easy to check if the dependencies in $F$ hold?

Okay as long as the dependencies can be checked in the same table.

Consider $R = (A, B, C)$, and $F = \{A \rightarrow B, B \rightarrow C\}$

1. Decompose into $R_1 = (A, B)$, and $R_2 = (A, C)$
   
   Lossless? Yes.

   But, makes it hard to check for $B \rightarrow C$
   
   *The data is in multiple tables.*

2. On the other hand, $R_1 = (A, B)$, and $R_2 = (B, C)$,
   
   is both lossless and dependency-preserving

   Really? What about $A \rightarrow C$?

   If we can check $A \rightarrow B$, and $B \rightarrow C$, $A \rightarrow C$ is implied.
Definition:
- Consider decomposition of $R$ into $R_1$, $..., R_n$.
- Let $F_i$ be the set of dependencies $F^+$ that include only attributes in $R_i$.

The decomposition is dependency preserving, if

$$(F_1 \cup F_2 \cup ... \cup F_n)^+ = F^+$$
Outline

- Mechanisms and definitions to work with FDs
  - Closures, candidate keys, canonical covers etc...
  - Armstrong axioms

- Decompositions
  - Loss-less decompositions, Dependency-preserving decompositions

- BCNF
  - How to achieve a BCNF schema

- BCNF may not preserve dependencies

- 3NF: Solves the above problem

- BCNF allows for redundancy

- 4NF: Solves the above problem
BCNF

- Given a relation schema $R$, and a set of functional dependencies $F$, if every FD, $A \rightarrow B$, is either:
  1. Trivial
  2. $A$ is a superkey of $R$

Then, $R$ is in **BCNF (Boyce-Codd Normal Form)**

- What if the schema is not in BCNF?
  - Decompose (split) the schema into two pieces.
  - Careful: you want the decomposition to be lossless
Achieving BCNF Schemas

For all dependencies $A \rightarrow B$ in $F+$, check if $A$ is a superkey
   By using attribute closure

If not, then
   Choose a dependency in $F+$ that breaks the BCNF rules, say $A \rightarrow B$
   Create $R1 = A \ B$
   Create $R2 = A \ (R - B - A)$
   Note that: $R1 \cap R2 = A$ and $A \rightarrow AB (= R1)$, so this is lossless decomposition

Repeat for $R1$, and $R2$
   By defining $F1+$ to be all dependencies in $F$ that contain only attributes in $R1$
   Similarly $F2+$
Example 1

\[ R = (A, B, C) \]
\[ F = \{ A \rightarrow B, B \rightarrow C \} \]
Candidate keys = \{A\}
BCNF = No. B \rightarrow C violates.

B \rightarrow C

R1 = (B, C)
\[ F1 = \{ B \rightarrow C \} \]
Candidate keys = \{B\}
BCNF = true

R2 = (A, B)
\[ F2 = \{ A \rightarrow B \} \]
Candidate keys = \{A\}
BCNF = true
Example 2-1

\[ R = (A, B, C, D, E) \]
\[ F = \{A \rightarrow B, BC \rightarrow D\} \]
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\} etc…

**Dependency preservation ???**

We can check:
- \(A \rightarrow B\) (R1), \(AC \rightarrow D\) (R3),
  but we lost \(BC \rightarrow D\)
So this is not a dependency-preserving decomposition
Example 2-2

R = (A, B, C, D, E)
F = \{A \rightarrow B, BC \rightarrow D\}
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\} etc…

BC \rightarrow D

R1 = (B, C, D)
F1 = \{BC \rightarrow D\}
Candidate keys = \{BC\}
BCNF = true

R2 = (B, C, A, E)
F2 = \{A \rightarrow B\}
Candidate keys = \{ACE\}
BCNF = false (A \rightarrow B)

A \rightarrow B

R3 = (A, B)
F3 = \{A \rightarrow B\}
Candidate keys = \{A\}
BCNF = true

R4 = (A, C, E)
F4 = \{\} [only trivial]
Candidate keys = \{ACE\}
BCNF = true

Dependency preservation ???
We can check:
BC \rightarrow D (R1), A \rightarrow B (R3),
Dependency-preserving decomposition
**Example 3**

\[ R = (A, B, C, D, E, H) \]
\[ F = \{ A \rightarrow BC, E \rightarrow HA \} \]
Candidate keys = \{DE\}

BCNF = Violated by \{A \rightarrow BC\} etc…

- **A \rightarrow BC**
  - \( R_1 = (A, B, C) \)
  - \( F_1 = \{ A \rightarrow BC \} \)
  - Candidate keys = \{A\}
  - BCNF = true

- **E \rightarrow HA**
  - \( R_3 = (E, H, A) \)
  - \( F_3 = \{ E \rightarrow HA \} \)
  - Candidate keys = \{E\}
  - BCNF = true

- **R_2 = (A, D, E, H)\]
  - \( F_2 = \{ E \rightarrow HA \} \)
  - Candidate keys = \{DE\}
  - BCNF = false (E \rightarrow HA)

- **R_4 = (ED)\]
  - \( F_4 = \{ \} \) [[ only trivial ]]\]
  - Candidate keys = \{DE\}
  - BCNF = true

Dependency preservation ???
We can check:

- A \rightarrow BC (R1), E \rightarrow HA (R3),
- Dependency-preserving decomposition
Mechanisms and definitions to work with FDs
- Closures, candidate keys, canonical covers etc...
- Armstrong axioms

Decompositions
- Loss-less decompositions, Dependency-preserving decompositions

BCNF
- How to achieve a BCNF schema

BCNF may not preserve dependencies

3NF: Solves the above problem

BCNF allows for redundancy

4NF: Solves the above problem
BCNF may not preserve dependencies

- $R = \{J, K, L\}$
- $F = \{JK \rightarrow L, L \rightarrow K\}$

- Two candidate keys = JK and JL

- $R$ is not in BCNF

- Any decomposition of $R$ will fail to preserve $JK \rightarrow L$

- This implies that testing for $JK \rightarrow L$ requires a join
BCNF may not preserve dependencies

- Not always possible to find a dependency-preserving decomposition that is in BCNF.

- PTIME to determine if there exists a dependency-preserving decomposition in BCNF
  - in size of F

- NP-Hard to find one if it exists

- Better results exist if F satisfies certain properties
Outline

- Mechanisms and definitions to work with FDs
  - Closures, candidate keys, canonical covers etc...
  - Armstrong axioms
- Decompositions
  - Loss-less decompositions, Dependency-preserving decompositions
- BCNF
  - How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem