Unit 9
Logical Database Design
With Normalization 2
So far, our treatment was not algorithmic and we just looked at an interesting case exploring within the context of that case 3 issues.

1. Avoiding (some) redundancies by converting tables to 3NF (and sometimes getting BCNF)
2. Preserving dependencies/constraints by making sure that dependencies (business rules) can be easily checked and enforced
3. Making sure that the decomposition of tables to obtain tables in better form does not cause us to lose information (lossless join) decomposition
Now: To Algorithmic Techniques

- While we looked at examples to build up an intuition, we did not have an algorithmic procedure to deal with the design issues.
- We now continue with building up intuition and actually learning an algorithmic procedure.
- This is the procedure you will use in the course on all the questions you will be asked to answer.
- So the drawings we had are good to know, we will not use them for normalization.
Closures Of Sets Of Attributes (Column Names)

Closure of a set of attributes is an easy to use but extremely powerful tool for everything that follows.

“On the way” we may review some concepts.

We return to our old example, in which we are given a table with three columns (attributes):

• Employee (E, for short, meaning really the SSN of the employee)
• Grade (G, for short)
• Salary (S, for short)

Satisfies:
1. E → G
2. G → S

We would like to find all the keys of this table.

A key is a minimal set of attributes, such that the values of these attributes, “force” some values for all the other attributes.
Closures Of Sets Of Attributes

In general, we have a concept of a **the closure of a set of attributes**

Let X be a set of attributes, then X⁺ is the set of all attributes, whose values are forced by the values of X

In our example

- E⁺ = EGS (because given E we have the value of G and then because we have the value for G we have the value for E)
- G⁺ = GS
- S⁺ = S

This is interesting because we have just showed that E is a key

And here we could also figure out that this is the only key, as GS⁺ = GS, so we will never get E unless we already have it

Note that GS⁺ really means (GS)⁺ and not G(S)⁺
Computing a Closure: An Example

- Our table is ABCDE
- Our only functional dependency (FD) is BC → D
  - This means: any tuples that are equal on both B and on C must be equal on D also
- We look at all the tuples of the table in which ABC has a specific fixed value, that is all the values of A are the same, all the values of B are the same and all the values of C are the same
  - We discuss soon why this is interesting
- What other columns from D and E have specific fixed values for the set of tuples we are considering?

- D has to have a specific fixed value
- E does not have to have a specific fixed value
**Computing Closures Of Sets Of Attributes**

There is a very simple algorithm to compute $X^+$

1. Let $Y = X$
2. Whenever there is an FD, say $V \rightarrow W$, such that
   1. $V \subseteq Y$, and
   2. $W - Y$ is not empty
   add $W - Y$ to $Y$
3. At termination $Y = X^+$

The algorithm is very efficient

Each time we look at all the functional dependencies
  - Either we can apply at least one functional dependency and make $Y$ bigger (the biggest it can be are all attributes), or
  - We are finished
Example

Let \( R = ABCDEGHIJK \)

Given FDs:
1. \( K \rightarrow BG \)
2. \( A \rightarrow DE \)
3. \( H \rightarrow AI \)
4. \( B \rightarrow D \)
5. \( J \rightarrow IH \)
6. \( C \rightarrow K \)
7. \( I \rightarrow J \)

We will compute: \( ABC^+ \)
1. We start with \( ABC^+ = ABC \)
2. Using FD number 2, we now have: \( ABC^+ = ABCDE \)
3. Using FD number 6, we now have \( ABC^+ = ABCDEK \)
4. Using FD number 1, we now have \( ABC^+ = ABCDEK\)

No FD can be applied productively anymore and we are done
The notion of an FD allows us to formally define keys.

Given R (relation schema which is always denoted by its set of attributes), satisfying a set of FDs, a set of attributes X of R is a key, if and only if:

- \( X^+ = R \).
- For any \( Y \subseteq X \) such that \( Y \neq X \), we have \( Y^+ \neq R \).

Note that if R does not satisfy any (nontrivial) FDs, then R is the only key of R.

“Trivial” means \( P \rightarrow Q \) and \( Q \subseteq P \): we saying something that is always true and not interesting.

Example, \( AB \rightarrow A \) is always true and does not say anything interesting.

Example, if a table is \( R(\text{Name}, \text{Surname}) \) without any functional dependencies, then its key is just the pair \( (\text{Name}, \text{Surname}) \).
If we apply our algorithm to the EGS example given earlier, we can now just compute that E was (the only) key by checking all the subsets of \{E, G, S\}.

Of course, in general, our algorithm is not efficient, but in practice what we do will be very efficient (most of the times).
Example

Let R = ABCDEKGIJ

Given FDs:

1. K → BG
2. A → DE
3. H → AI
4. B → D
5. J → IH
6. C → K
7. I → J

Then

• ABCH⁺ = ABCDEGHIJK
• And ABCH is a key or maybe contains a key as a proper subset
• We could check whether ABCH is a key by computing ABC⁺, ABH⁺, ACH⁺, BCH⁺ and showing that none of them is ABCDEGHIJK
Another Example: Airline Scheduling

- We have a table PFDT, where
  - PILOT
  - FLIGHT NUMBER
  - DATE
  - SCHEDULED_TIME_of_DEPARTURE

- The table satisfies the FDs:
  - F → T
  - PDT → F
  - FD → P
Computing Keys

- We will compute all the keys of the table
- In general, this will be an exponential-time algorithm in the size of the problem
- But there will be useful heuristic making this problem tractable in practice
- We will introduce some heuristics here and additional ones later

We note that if some subset of attributes is a key, then no proper superset of it can be a key as it would not be minimal and would have superfluous attributes
There is a natural structure (technically a lattice) to all the nonempty subsets of attributes.

I will draw the lattice here, in practice this is not done:
  - Not necessary and too big.

We will look at all the non-empty subsets of attributes.

There are 15 of them: $2^4 - 1$.

The structure is clear from the drawing.
Lattice Of Nonempty Subsets
Keys Of PFDT

- The algorithm proceeds from bottom up
- We first try all potential 1-attribute keys, by examining all 1-attribute sets of attributes
  - $P^+ = P$
  - $F^+ = FT$
  - $D^+ = D$
  - $T^+ = T$

There are no 1-attribute keys

- Note, that the it is impossible for a key to have both $F$ and $T$
  - Because if $F$ is in a key, $T$ will be automatically determined as it is included in the closure of $F$
- Therefore, we can prune our lattice
Pruned Lattice
Keys Of PFDT

We try all potential 2-attribute keys

- \( PF^+ = PFT \)
- \( PD^+ = PD \)
- \( PT^+ = PT \)
- \( FD^+ = FDPT \)
- \( DT^+ = DT \)

There is one 2-attribute key: FD

We can mark the lattice

We can prune the lattice
Marked And Pruned Lattice

- The key we found is marked with red
- Some nodes can be removed
We try all potential 3-attribute keys

- $PDT^+ = PDT_F$

There is one 3-attribute key: PDT
Final Lattice
We Only Care About The Keys

We could have removed some nodes, but we did not need to do that as we found all the possible keys.
Finding A Decomposition

Next, we will discuss by means of an example how to decompose a table into tables, such that

1. The decomposition is lossless join
2. Dependencies are preserved
3. Each resulting table is in 3NF

Although this will be described using an example, the example will be sufficiently general so that the general procedure will be covered
The EmToPrHoSkLoRo Table

- The table deals with employees who use tools on projects and work a certain number of hours per week
- An employee may work in various locations and has a variety of skills
- All employees having a certain skill and working in a certain location meet in a specified room once a week

The attributes of the table are:
- Em: Employee
- To: Tool
- Pr: Project
- Ho: Hours per week
- Sk: Skill
- Lo: Location
- Ro: Room for meeting
The table deals with employees who use tools on projects and work a certain number of hours per week.

An employee may work in various locations and has a variety of skills.

All employees having a certain skill and working in a certain location meet in a specified room once a week.

The table satisfies the following FDs:

- Each employee uses a single tool: Em → To
- Each employee works on a single project: Em → Pr
- Each tool can be used on a single project only: To → Pr
- An employee uses each tool for the same number of hours each week: EmTo → Ho
- All the employees working in a location having a certain skill always work in the same room (in that location): SkLo → Ro
- Each room is in one location only: Ro → Lo
## Sample Instance: Many Redundancies

<table>
<thead>
<tr>
<th>Em</th>
<th>To</th>
<th>Pr</th>
<th>Ho</th>
<th>Sk</th>
<th>Lo</th>
<th>Ro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Writer</td>
<td>Boston</td>
<td>102</td>
</tr>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Writer</td>
<td>Buffalo</td>
<td>103</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Clerk</td>
<td>New York</td>
<td>104</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Editor</td>
<td>New York</td>
<td>105</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Economist</td>
<td>New York</td>
<td>106</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Analyst</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Analyst</td>
<td>Buffalo</td>
<td>108</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Albany</td>
<td>109</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Trenton</td>
<td>110</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
</tbody>
</table>
Our FDs

1. \( \text{Em} \rightarrow \text{To} \)
2. \( \text{Em} \rightarrow \text{Pr} \)
3. \( \text{To} \rightarrow \text{Pr} \)
4. \( \text{EmTo} \rightarrow \text{Ho} \)
5. \( \text{SkLo} \rightarrow \text{Ro} \)
6. \( \text{Ro} \rightarrow \text{Lo} \)

What should we do with this drawing? I do not know. We need an algorithm.

We know how to find keys (we will actually do it later) and we can figure that \( \text{EmSkLo} \) could serve as the primary key, so we could draw using the appropriate colors.

But note that there for FD number 4, the left hand side contains an attribute from the key and an attribute from outside the key, so I used a new color.

Let’s forget for now that I have told you what the primary key was, we will find it later.
1: Getting A Minimal Cover

We need to “simplify” our set of FDs to bring it into a “nicer” form, so called minimal cover or (sometimes called also canonical cover)

But, of course, the “business rule” power has to be the same as we need to enforce the same business rules

The algorithm for this will be covered later, it is very important

The end result is:
1. Em → ToHo
2. To → Pr
3. SkLo → Ro
4. Ro → Lo

From these we will build our tables directly, but just for fun, we can look at a drawing
2: Creating Tables From a Minimal Cover

Create a table for each functional dependency

We obtain the tables:

1. EmToHo
2. ToPr
3. SkLoRo
4. LoRo
LoRo is a subset of SkLoRo, so we remove it.
We obtain the tables:
1. EmToHo
2. ToPr
3. SkLoRo
4: Ensuring The Storage Of The Global Key
(Of The Original Table)

- We need to have a table containing the global key
- Perhaps one of our tables contain such a key
- So we check if any of them already contains a key of EmToPrHoSkLoRo:
  1. EmToHo $EmToHo^+ = EmToHoPr$, does not contain a key
  2. ToPr $ToPr^+ = ToPr$, does not contain a key
  3. SkLoRo $SkLoRo^+ = SkLoRo$, does not contain a key

- We need to add a table whose attributes form a global key
Finding Keys Using a Good Heuristic

Let us list the FDs again (or could have worked with the minimal cover, does not matter):

- Em $\rightarrow$ To
- Em $\rightarrow$ Pr
- To $\rightarrow$ Pr
- EmTo $\rightarrow$ Ho
- SkLo $\rightarrow$ Ro
- Ro $\rightarrow$ Lo

We can classify the attributes into 4 classes:

1. Appearing on both sides of FDs; here To, Lo, Ro.
2. Appearing on left sides only; here Em, Sk.
3. Appearing on right sides only; here Pr, Ho.
4. Not appearing in FDs; here none.
Finding Keys

Facts:

- Attributes of class 2 and 4 must appear in every key
- Attributes of class 3 do not appear in any key
- Attributes of class 1 may or may not appear in keys

An algorithm for finding keys relies on these facts

- Unfortunately, in the worst case, exponential in the number of attributes

Start with the attributes in classes 2 and 4, add as needed (going bottom up) attributes in class 1, and ignore attributes in class 3
Finding Keys

- In our example, therefore, every key must contain EmSk.
- To see, which attributes, if any have to be added, we compute which attributes are determined by EmSk.
- We obtain:
  - $\text{EmSk}^+ = \text{EmToPrHoSk}$
- Therefore Lo and Ro are missing.
- It is easy to see that the table has two keys:
  - EmSkLo
  - EmSkRo
Finding Keys

Although not required strictly by the algorithm (which does not mind decomposing a table in 3NF into tables in 3NF) we can check if the original table was in 3NF.

We conclude that the original table is not in 3NF, as for instance, $To \rightarrow Pr$ is a transitive dependency and therefore not permitted for 3NF.
None of the tables contains either EmSkLo or EmSkRo. Therefore, one more table needs to be added. We have 2 choices for the final decomposition:

1. EmToHo; satisfying Em → ToHo; primary key: Em
2. ToPr; satisfying To → Pr; primary key To
3. SkLoRo; satisfying SkLo → Ro and Ro → Lo; primary key SkLo or SkRo
4. EmSkLo; not satisfying anything; primary key EmSkLo

or

1. EmToHo; satisfying Em → ToHo; primary key: Em
2. ToPr; satisfying To → Pr; primary key To
3. SkLoRo; satisfying SkLo → Ro and Ro → Lo; primary key SkLo or SkRo
4. EmSkRo; not satisfying anything; primary key SkRO

We have completed our process and got a decomposition with the properties we needed; actually more than one
A Decomposition

<table>
<thead>
<tr>
<th>Em</th>
<th>Sk</th>
<th>Lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Clerk</td>
<td>Boston</td>
</tr>
<tr>
<td>Mary</td>
<td>Writer</td>
<td>Boston</td>
</tr>
<tr>
<td>Mary</td>
<td>Writer</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Fang</td>
<td>Clerk</td>
<td>New York</td>
</tr>
<tr>
<td>Fang</td>
<td>Editor</td>
<td>New York</td>
</tr>
<tr>
<td>Fang</td>
<td>Economist</td>
<td>New York</td>
</tr>
<tr>
<td>Fang</td>
<td>Economist</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Analyst</td>
<td>Boston</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Analyst</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>Boston</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>Albany</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>Trenton</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Economist</td>
<td>Buffalo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Em</th>
<th>To</th>
<th>Ho</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>20</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>30</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sk</th>
<th>Lo</th>
<th>Ro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Writer</td>
<td>Boston</td>
<td>102</td>
</tr>
<tr>
<td>Writer</td>
<td>Buffalo</td>
<td>103</td>
</tr>
<tr>
<td>Clerk</td>
<td>New York</td>
<td>104</td>
</tr>
<tr>
<td>Editor</td>
<td>New York</td>
<td>105</td>
</tr>
<tr>
<td>Economist</td>
<td>New York</td>
<td>106</td>
</tr>
<tr>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Analyst</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Analyst</td>
<td>Buffalo</td>
<td>108</td>
</tr>
<tr>
<td>Clerk</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Clerk</td>
<td>Albany</td>
<td>109</td>
</tr>
<tr>
<td>Clerk</td>
<td>Trenton</td>
<td>110</td>
</tr>
</tbody>
</table>

© 2014 Zvi M. Kedem
# A Decomposition

<table>
<thead>
<tr>
<th>Em</th>
<th>Sk</th>
<th>Ro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Clerk</td>
<td>101</td>
</tr>
<tr>
<td>Mary</td>
<td>Writer</td>
<td>102</td>
</tr>
<tr>
<td>Mary</td>
<td>Writer</td>
<td>103</td>
</tr>
<tr>
<td>Fang</td>
<td>Clerk</td>
<td>104</td>
</tr>
<tr>
<td>Fang</td>
<td>Editor</td>
<td>105</td>
</tr>
<tr>
<td>Fang</td>
<td>Economist</td>
<td>106</td>
</tr>
<tr>
<td>Fang</td>
<td>Economist</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Analyst</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Analyst</td>
<td>108</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>109</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Clerk</td>
<td>110</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Economist</td>
<td>107</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sk</th>
<th>Lo</th>
<th>Ro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Writer</td>
<td>Boston</td>
<td>102</td>
</tr>
<tr>
<td>Writer</td>
<td>Buffalo</td>
<td>103</td>
</tr>
<tr>
<td>Clerk</td>
<td>New York</td>
<td>104</td>
</tr>
<tr>
<td>Editor</td>
<td>New York</td>
<td>105</td>
</tr>
<tr>
<td>Economist</td>
<td>New York</td>
<td>106</td>
</tr>
<tr>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Analyst</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Analyst</td>
<td>Buffalo</td>
<td>108</td>
</tr>
<tr>
<td>Clerk</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Clerk</td>
<td>Albany</td>
<td>109</td>
</tr>
<tr>
<td>Clerk</td>
<td>Trenton</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Em</th>
<th>To</th>
<th>Ho</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>20</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>30</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To</th>
<th>Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pen</td>
<td>Research</td>
</tr>
<tr>
<td>Oracle</td>
<td>Database</td>
</tr>
</tbody>
</table>
**Properties Of The Decomposition**

- The table on the left listed the values of the key of the original table.
- Each row corresponded to a row of the original table.
- The other tables had rows that could be “glued” to the “key” table based on the given business rules and thus reconstruct the original table.
- All the tables are in 3NF.
Computing Minimal Cover

What remains to be done is to learn how to start with a set of FDs and to “reduce” them to a “clean” set with equivalent constraints power.

This “clean” set is a minimal cover.

So we need to learn how to do that next.

We need first to understand better some properties of FDs.
To Remind: Functional Dependencies

Generally, if X and Y are sets of attributes, then $X \rightarrow Y$ means:

Any two tuples (rows) that are equal on (the vector of attributes) X

are also

equal on (the vector of attributes) Y

Note that this generalizes the concept of a key (UNIQUE, PRIMARY KEY)

• We do not insist that X determines everything

• For instance we say that any two tuples that are equal on G are equal on S, but we do not say that any two tuples that are equal on G are “completely” equal
An Example

Functional dependencies are properties of a schema, that is, all permitted instances.

For practice, we will examine an instance:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>f1</td>
<td>g1</td>
<td>h1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
<td>e2</td>
<td>f2</td>
<td>g1</td>
<td>h1</td>
</tr>
<tr>
<td>3</td>
<td>a2</td>
<td>b2</td>
<td>c3</td>
<td>d3</td>
<td>e3</td>
<td>f3</td>
<td>g1</td>
<td>h2</td>
</tr>
<tr>
<td>4</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>f4</td>
<td>g2</td>
<td>h3</td>
</tr>
<tr>
<td>5</td>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e4</td>
<td>f5</td>
<td>g2</td>
<td>h4</td>
</tr>
<tr>
<td>6</td>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d2</td>
<td>e5</td>
<td>f6</td>
<td>g2</td>
<td>h3</td>
</tr>
</tbody>
</table>

1. A → C
2. AB → C
3. E → CD
4. D → B
5. F → ABC
6. H → G
7. H → GE
8. HGE → GE
Let us look at another example first

Consider some table talking about employees in which there are three columns:
1. Grade
2. Bonus
3. Salary

Consider now two possible FDs (functional dependencies)
1. Grade → Bonus
2. Grade → Bonus Salary

FD (2) is more restrictive, fewer relations will satisfy FD (2) than satisfy FD (1)
• So FD (2) is stronger
• Every relation that satisfies FD (2), must satisfy FD (1)
• And we know this just because \{Bonus\} is a proper subset of \{Bonus, Salary\}
Relative Power Of Some FDs
H → G vs. H → GE

- An important note: H → GE is always at least as powerful as H → G
  that is

- If a relation satisfies H → GE it must satisfy H → G

- What we are really saying is that if GE = f(H), then of course G = f(H)

- An informal way of saying this: if being equal on H forces to be equal on GE, then of course there is equality just on G

- More generally, if X, Y, Z, are sets of attributes and Z ⊆ Y; then if X → Y is true than X → Z is true
Relative Power Of Some FDs

A → C vs. AB → C

Let us look at another example first.

Consider some table talking about employees in which there are three columns:
1. Grade
2. Location
3. Salary

Consider now two possible FDs
1. Grade → Salary
2. Grade Location → Salary

FD (2) is less restrictive, more relations will satisfy FD (2) than satisfy FD (1)
• So FD (1) is stronger
• Every relation that satisfies FD (1), must satisfy FD (2)
• And we know this just because \{Grade\} is a proper subset of \{Grade, Salary\}
Relative Power Of Some FDs
\[ A \rightarrow C \text{ vs. } AB \rightarrow C \]

An important note: \( A \rightarrow C \) is always at least as powerful as \( AB \rightarrow C \)

that is

If a relation satisfies \( A \rightarrow C \) it must satisfy \( AB \rightarrow C \)

What we are really saying is that if \( C = f(A) \), then of course \( C = f(A,B) \)

An informal way of saying this: if just being equal on A forces to be equal on C, then if we in addition know that there is equality on B also, of course it is still true that there is equality on C

More generally, if \( X, Y, Z \), are sets of attributes and \( X \subseteq Y \); then if \( X \rightarrow Z \) is true than \( Y \rightarrow Z \) is true
Trivial FDs

- An FD $X \rightarrow Y$, where $X$ and $Y$ are sets of attributes is trivial if and only if $Y \subseteq X$

  (Such an FD gives no constraints, as it is always satisfied, which is easy to prove)

Example
- Grade, Salary $\rightarrow$ Grade is trivial

A trivial FD does not provide any constraints
Every relations that contains columns Grade and Salary will satisfy this FD: Grade, Salary $\rightarrow$ Grade
Decomposition and Union of some FDs

An FD \( X \rightarrow A_1 A_2 \ldots A_m \), where \( A_i \)'s are individual attributes

is equivalent to

the set of FDs:
\[
\begin{align*}
X & \rightarrow A_1 \\
X & \rightarrow A_2 \\
& \vdots \\
X & \rightarrow A_m
\end{align*}
\]

Example

FirstName LastName \( \rightarrow \) Address Salary

is equivalent to the set of the two FDs:

Firstname LastName \( \rightarrow \) Address
Firstname LastName \( \rightarrow \) Salary
Logical implications of FDs

It will be important to us to determine if a given set of FDs forces some other FDs to be true.

Consider again the EGS relation.

Which FDs are satisfied?
• E → G, G → S, E → S are all true in the real world.

If the real world tells you only:
• E → G and G → S

Can you deduce on your own (and is it even always true?), without understanding the semantics of the application, that
• E → S?
Logical implications of FDs

Yes, by simple logical argument: transitivity
1. Take any (set of) tuples that are equal on E
2. Then given E → G we know that they are equal on G
3. Then given G → S we know that they are equal on S
4. So we have shown that E → S must hold

We say that E → G, G → S \textit{logically imply} E → S and we write

\[
E \rightarrow G, \ G \rightarrow S \models E \rightarrow S
\]

This means:
If a relation satisfies E → G and G → S,
then
It must satisfy E → S
Logical implications of FDs

- If the real world tells you only:
  - $E \rightarrow G$ and $E \rightarrow S$,

- Can you deduce on your own, without understanding the application that
  - $G \rightarrow S$

- No, because of a counterexample:

<table>
<thead>
<tr>
<th>EGS</th>
<th>E</th>
<th>G</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>A</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Beta</td>
<td>A</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

- This relation satisfies $E \rightarrow G$ and $E \rightarrow S$, but violates $G \rightarrow S$

- For intuitive explanation, think: $G$ means Height and $S$ means Weight
Consider a relation EGS for which the three constraints $E \rightarrow G$, $G \rightarrow S$, and $E \rightarrow S$ must all be obeyed!

It is enough to make sure that the two constraints $E \rightarrow G$ and $G \rightarrow S$ are not violated!

It is not enough to make sure that the two constraints $E \rightarrow G$ and $E \rightarrow S$ are not violated!

But what to do in general, large, complex cases?
To Remind: Closures Of Sets Of Attributes

We consider some relation schema, which is a set of attributes, R (say EGS, which could also write as R(EGS))

A set F of FDS for this schema (say E → G and G → S)

We take some $X \subseteq R$ (Say just the attribute E)

We ask if two tuples are equal on $X$, what is the largest set of attributes on which they must be equal

We call this set the closure of $X$ with respect to $F$ and denote it by $X_F^+$ (in our case $E_{F^+} = EGS$ and $S_{F^+} = S$, as is easily seen)

If it is understood what $F$ is, we can write just $X^+$
Towards A Minimal Cover

- This form will be based on trying to store a “concise” representation of FDs
- We will try to find a “small” number of “small” relation schemas that are sufficient to maintain the FDs
- The core of this will be to find “concise” description of FDs
  - Example: in ESG, E → S was not needed
- We will compute a minimal cover for a set of FDs
- The basic idea, simplification of a set of FDs by
  - Combining FDs when possible
  - Getting rid of unnecessary attributes
- We will start with examples to introduce the concepts and the tools
**Union Rule: Combining Right Hand Sides (RHSs)**

- \( F = \{ AB \rightarrow C, AB \rightarrow D \} \)

  *is equivalent to*

- \( H = \{ AB \rightarrow CD \} \)

- We have discussed this rule before
- Intuitively clear
- Formally we need to prove 2 things
  - \( F \models H \) is true; we do this (as we know) by showing that \( AB_F^+ \) contains CD; easy exercise
  - \( H \models F \) is true; we do this (as we know) by showing that \( AB_H^+ \) contains C and \( AB_H^+ \) contains D; easy exercise

- Note: you *cannot* combine LHSs based on equality of RHS and get an equivalent set of FDS
  - \( F = \{ A \rightarrow C, B \rightarrow C \} \) *is stronger than* \( H = \{ AB \rightarrow C \} \)
**Union Rule: Combining Right Hand Sides (RHSs)**

**Stated formally:**

F = \{ X \rightarrow Y, X \rightarrow Z \} \textit{is as powerful as} H = \{ X \rightarrow YZ \}

**Easy proof, we omit**
Relative Power Of FDs: Left Hand Side (LHS)

F = \{ AB \rightarrow C \}

is weaker than

H = \{ A \rightarrow C \}

We have discussed this rule before when we started talking about FDs.

Intuitively clear: in F, if we assume more (equality on both A and B) to conclude something (equality on C) than our FD is applicable in fewer case (does not work if we have equality is true on B’s but not on C’S) and therefore F is weaker than H.

Formally we need to prove two things:

• F \models H is false; we do this (as we know) by showing that A_{F^+} does not contain C; easy exercise.
• H \models F is true; we do this (as we know) by showing that AB_{H^+} contains C; easy exercise.
Relative Power Of FDs: Left Hand Side (LHS)

- Stated formally:
  \[ F = \{ XB \rightarrow Y \} \text{ is weaker than } H = \{ X \rightarrow Y \}, \text{ (if } B \not\subset X) \]

- Easy proof, we omit

- Can state more generally, replacing B by a set of attributes, but we do not need this
Relative Power Of FDs: Right Hand Side (RHS)

\[ F = \{ A \rightarrow BC \} \]

is stronger than

\[ H = \{ A \rightarrow B \} \]

Intuitively clear: in H, we deduce less from the same assumption, equality on A’s

Formally we need to prove two things

- \( F \models H \) is true; we do this (as we know) by showing that \( A_F^+ \) contains B; easy exercise
- \( H \models F \) is false; we do this (as we know) by showing that \( A_H^+ \) does not contain C; easy exercise
Stated formally:

\[ F = \{ X \rightarrow YC \} \text{ is stronger than } H = \{ X \rightarrow Y \}, \text{ (if } C \not\in Y \text{ and } C \not\in X) \]

Easy proof, we omit

Can state more generally, replacing C by a set of attributes, but we do not need this
Simplifying Sets Of FDs

At various stages of the algorithm we will have
- An “old” set of FDs
- A “new” set of FDs

The two sets will not vary by “very much”

We will indicate the parts that do not change by . . .

Of course, as we are dealing with sets, the order of the FDs in the set does not matter
Simplifying Set Of FDs
By Using The Union Rule

X, Y, Z are sets of attributes

Let F be:

...  
X → Y  
X → Z

Then, F is equivalent to the following H:

...  
X → YZ
**Simplify Set Of FDS**
*By Simplifying LHS*

Let $X, Y$ be sets of attributes and $B$ a single attribute not in $X$.

Let $F$ be:

\[
\cdots \\
XB \rightarrow Y
\]

Let $H$ be:

\[
\cdots \\
X \rightarrow Y
\]

Then if $F \models X \rightarrow Y$ holds, then we can replace $F$ by $H$ without changing the “power” of $F$.

We do this by showing that $X_F^+$ contains $Y$.

- $H$ could only be stronger, but we are proving it is not actually stronger, but equivalent.
Simplify Set Of FDS
By Simplifying LHS

- H can only be stronger than F, as we have replaced a weaker FD by a stronger FD
- But if we F |= H holds, this “local” change does not change the overall power
- Example below
- Replace
  - • AB → C
  - • A → B
  by
  - • A → C
  - • A → B
Simplify Set Of FDS
By Simplifying RHS

- Le X, Y are sets of attributes and C a single attribute not in Y
- Let F be:
  
  ...  
  X → YC  
  ...  

- Let H be:
  
  ...  
  X → Y  
  ...  

- Then if H |= X → YC holds, then we can replace F by H without changing the “power” of F
- We do this by showing that $X_H^+$ contains YC
  - H could only be weaker, but we are proving it is not actually weaker, but equivalent
Simplify Set Of FDS
By Simplifying RHS

- H can only be weaker than F, as we have replaced a stronger FD by a weaker FD
- But if we H |= F holds, this “local” change does not change the overall power
- Example below

Replace
- A → BC
- B → C

by
- A → B
- B → C
Minimal Cover

Given a set of FDs $F$, find a set of FDs $F_m$, that is (in a sense we formally define later) minimal.

Algorithm:
1. Start with $F$
2. Remove all trivial functional dependencies
3. Repeatedly apply (in whatever order you like), until no changes are possible
   - Union Simplification (it is better to do it as soon as possible, whenever possible)
   - RHS Simplification
   - LHS Simplification
4. What you get is a minimal cover

We proceed through a largish example to exercise all possibilities
The EmToPrHoSkLoRo Relation

- The relation deals with employees who use tools on projects and work a certain number of hours per week.
- An employee may work in various locations and has a variety of skills.
- All employees having a certain skill and working in a certain location meet in a specified room once a week.

The attributes of the relation are:

- Em: Employee
- To: Tool
- Pr: Project
- Ho: Hours per week
- Sk: Skill
- Lo: Location
- Ro: Room for meeting
The relation deals with employees who use tools on projects and work a certain number of hours per week.

An employee may work in various locations and has a variety of skills.

All employees having a certain skill and working in a certain location meet in a specified room once a week.

The relation satisfies the following FDs:

- Each employee uses a single tool: \( \text{Em} \rightarrow \text{To} \)
- Each employee works on a single project: \( \text{Em} \rightarrow \text{Pr} \)
- Each tool can be used on a single project only: \( \text{To} \rightarrow \text{Pr} \)
- An employee uses each tool for the same number of hours each week: \( \text{EmTo} \rightarrow \text{Ho} \)
- All the employees working in a location having a certain skill always work in the same room (in that location): \( \text{SkLo} \rightarrow \text{Ro} \)
- Each room is in one location only: \( \text{Ro} \rightarrow \text{Lo} \)
## Sample Instance

<table>
<thead>
<tr>
<th>Em</th>
<th>To</th>
<th>Pr</th>
<th>Ho</th>
<th>Sk</th>
<th>Lo</th>
<th>Ro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Writer</td>
<td>Boston</td>
<td>102</td>
</tr>
<tr>
<td>Mary</td>
<td>Pen</td>
<td>Research</td>
<td>20</td>
<td>Writer</td>
<td>Buffalo</td>
<td>103</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Clerk</td>
<td>New York</td>
<td>104</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Editor</td>
<td>New York</td>
<td>105</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Economist</td>
<td>New York</td>
<td>106</td>
</tr>
<tr>
<td>Fang</td>
<td>Pen</td>
<td>Research</td>
<td>30</td>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Analyst</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Analyst</td>
<td>Buffalo</td>
<td>108</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Buffalo</td>
<td>107</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Boston</td>
<td>101</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Albany</td>
<td>109</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Clerk</td>
<td>Trenton</td>
<td>110</td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Oracle</td>
<td>Database</td>
<td>40</td>
<td>Economist</td>
<td>Buffalo</td>
<td>107</td>
</tr>
</tbody>
</table>
Our FDs

1. $\text{Em} \rightarrow \text{To}$
2. $\text{Em} \rightarrow \text{Pr}$
3. $\text{To} \rightarrow \text{Pr}$
4. $\text{EmTo} \rightarrow \text{Ho}$
5. $\text{SkLo} \rightarrow \text{Ro}$
6. $\text{Ro} \rightarrow \text{Lo}$
Run The Algorithm

Using the union rule, we combine RHS of 1 and 2, getting:

1. Em → ToPr
2. To → Pr
3. EmTo → Ho
4. SkLo → Ro
5. Ro → Lo
Run The Algorithm

☐ No RHS can be combined, so we check whether there are any redundant attributes.

☐ We start with FD 1, where we attempt to remove an attribute from RHS
  - We check whether we can remove To. This is possible if we can derive \( Em \rightarrow To \) using
    - \( Em \rightarrow Pr \)
    - \( To \rightarrow Pr \)
    - \( EmTo \rightarrow Ho \)
    - \( SkLo \rightarrow Ro \)
    - \( Ro \rightarrow Lo \)

  Computing the closure of \( Em \) using the above FDs gives us only \( EmPr \), so the attribute \( To \) must be kept.
Run The Algorithm

• We check whether we can remove Pr. This is possible if we can derive \( Em \rightarrow Pr \) using
  
  \[
  \begin{align*}
  Em & \rightarrow To \\
  To & \rightarrow Pr \\
  EmTo & \rightarrow Ho \\
  SkLo & \rightarrow Ro \\
  Ro & \rightarrow Lo
  \end{align*}
  \]

  Computing the closure of \( Em \) using the above FDs gives us \( EmToPrHo \), so the attribute \( Pr \) is redundant.
Run The Algorithm

We now have

1. Em → To
2. To → Pr
3. EmTo → Ho
4. SkLo → Ro
5. Ro → Lo

No RHS can be combined, so we continue attempting to remove redundant attributes. The next one is FD 3, where we attempt to remove an attribute from LHS

- We check if Em can be removed. This is possible if we can derive To → Ho using all the FDs. Computing the closure of To using the FDs gives ToPr, and therefore Em cannot be removed
- We check if To can be removed. This is possible if we can derive Em → Ho using all the FDs. Computing the closure of Em using the FDs gives EmToPrHo, and therefore To can be removed
Run The Algorithm

We now have
1. Em $\rightarrow$ To
2. To $\rightarrow$ Pr
3. Em $\rightarrow$ Ho
4. SkLo $\rightarrow$ Ro
5. Ro $\rightarrow$ Lo

We can now combine RHS of 1 and 3 and get
1. Em $\rightarrow$ ToHo
2. To $\rightarrow$ Pr
3. SkLo $\rightarrow$ Ro
4. Ro $\rightarrow$ Lo
Run The Algorithm

We now have

1. $\text{Em} \rightarrow \text{ToHo}$
2. $\text{To} \rightarrow \text{Pr}$
3. $\text{SkLo} \rightarrow \text{Ro}$
4. $\text{Ro} \rightarrow \text{Lo}$

No RHS can be combined, so we continue attempting to remove redundant attributes.

The first one is FD 1, where we attempt to remove an attribute from RHS

We check if To can be removed. This is possible if we can derive $\text{Em} \rightarrow \text{ToHo}$ using

$\text{Em} \rightarrow \text{Ho}$
$\text{To} \rightarrow \text{Pr}$
$\text{SkLo} \rightarrow \text{Ro}$
$\text{Ro} \rightarrow \text{Lo}$

$\text{Em}^+ \text{ is EmHo only so To cannot be removed}$
Run The Algorithm

We check if Ho can be removed. This is possible if we can derive $Em \rightarrow ToHo$ using

$Em \rightarrow To$
$To \rightarrow Pr$
$SkLo \rightarrow Ro$
$Ro \rightarrow Lo$

$Em^+$ is $EmToPr$ only so $Ho$ cannot be removed

The next one is FD 3, where we attempt to remove an attribute from LHS

We have actually checked this before and nothing has changed that could impact the decision so we will not do it again here.
Run The Algorithm

- Nothing else can be done
- Therefore we are done
- We have computed a minimal cover for the original set of FDs
Minimal Cover

A set of FDs, $F_m$, is a minimal cover for a set of FD $F$, if and only if

1. $F_m$ is minimal, that is
   1. No two FDs in it can be combined using the union rule
   2. No attribute can be removed from a RHS of any FD in $F_m$ without changing the power of $F_m$
   3. No attribute can be removed from a LHS of any FD in $F_m$ without changing the power of $F_m$

2. $F_m$ is equivalent in power to $F$

Note that there could be more than one minimal cover for $F$, as we have not specified the order of applying the simplification operations
How About EGS

- Applying to algorithm to EGS with
  1. $E \rightarrow G$
  2. $G \rightarrow S$
  3. $E \rightarrow S$

- Using the union rule, we combine 1 and 3 and get
  1. $E \rightarrow GS$
  2. $G \rightarrow S$

- Simplifying RHS of 1 (this is the only attribute we can remove), we get
  1. $E \rightarrow G$
  2. $G \rightarrow S$

- We automatically got the two “important” FDs!
An Algorithm For “An Almost”
3NF Lossless-Join Decomposition

Input: relation schema R and a set of FDs F
Output: almost-decomposition of R into R1, R2, …, Rn, each in 3NF

Algorithm
1. Produce $F_m$, a minimal cover for F
2. For each $X \rightarrow Y$ in $F_m$ create a new relation schema $XY$
3. For every new relation schema that is a subset (including being equal) of another new relation schema (that is the set of attributes is a subset of attributes of another schema or the two sets of attributes are equal) remove this relation schema (the “smaller” one or one of the equal ones); but if the two are equal, need to keep one of them
4. The set of the remaining relation schemas is an “almost final decomposition”
For our EmToPrHoSkLoRo example, we previously computed the following minimal cover:

1. Em → ToHo
2. To → Pr
3. SkLo → Ro
4. Ro → Lo
Creating Relations

Create a relation for each functional dependency

We obtain the relations:

1. EmToHo
2. ToPr
3. SkLoRo
4. LoRo
Removing Redundant Relations

LoRo is a subset of SkLoRo, so we remove it.

We obtain the relations:

1. EmToHo
2. ToPr
3. SkLoRo
How About EGS

The minimal cover was
1. \( E \rightarrow G \)
2. \( G \rightarrow S \)

Therefore the relations obtained were:
1. \( EG \)
2. \( GS \)

And this is exactly the decomposition we thought was best!
Assuring Storage Of A Global Key

- If no relation contains a key of the original relation, add a relation whose attributes form such a key

- Why do we need to do this?
  - Because otherwise we may not have a decomposition
  - Because otherwise the decomposition may not be lossless
Why It Is Necessary To Store A Global Key

Example

Consider the relation LnFn:
• Ln: Last Name
• Fn: First Name

There are no FDs

The relation has only one key:
• LnFn

Our algorithm (without the key included) produces no relations

A condition for a decomposition: Each attribute of R has to appear in at least one Ri

So we did not have a decomposition

But if we add the relation consisting of the attributes of the key
• We get LnFn (this is fine, because the original relations had no problems and was in a good form, actually in BCNF, which is always true when there are no (nontrivial) FDs)
Why It Is Necessary To Store A Global Key

Example

Consider the relation: LnFnVaSa:
• Ln: Last Name
• Fn: First Name
• Va: Vacation days per year
• Sa: Salary

The functional dependencies are:
• Ln → Va
• Fn → Sa

The relation has only one key
• LnFn

The relation is not in 3NF
• Ln → Va: Ln does not contain a key and Va is not in any key
• Fn → Sa: Fn does not contain a key and Sa is not in any key
Why It Is Necessary To Store A Global Key

Example

Our algorithm (without the key being included) will produce the decomposition

1. LnVa
2. FnSa

This is not a lossless-join decomposition

• In fact we do not know who the employees are (what are the valid pairs of LnFn)

So we decompose

1. LnVa with (primary) key Ln
2. FnSa with (primary) key Fn
3. LnFn with (primary) key LnFn
Assuring Storage Of A Global Key

- If no relation contains a key of the original relation, add a relation whose attributes form such a key.
- It is easy to test if a “new” relation contains a key of the original relation.
- Compute the closure of the relation with respect to all FDs (either original or minimal cover, it’s the same) and see if you get all the attributes of the original relation.
- If not, you need to find some key of the original relation.
- We have studied this before.
Applying The Algorithm to EGS

Applying the algorithm to EGS, we get our desired decomposition:

- EG
- GS

And the “new” relations are in BCNF too, though we guaranteed only 3NF!
Returning to Our Example

We pick the decomposition

1. EmToHo
2. ToPr
3. SkLoRo
4. EmSkLo

We have the minimal set of FDs of the simplest form (before any combinations)

1. Em → ToHo
2. To → Pr
3. SkLo → Ro
4. Ro → Lo
Returning to Our Example

Everything can be described as follows:

The relations, their keys, and FDs that need to be explicitly mentioned are:

1. EmToHo key: Em
2. ToPr key: To
3. SkLoRo key: SkLo, key SkRo, and functional dependency Ro → Lo
4. EmSkLo key: EmSkLo

In general, when you decompose as we did, a relation may have several keys and satisfy several FDs that do not follow from simply knowing keys.

In the example above there was one relation that had such an FD, which made is automatically not a BCNF relation (but by our construction a 3NF relation)
How are we going to express in SQL what we have learned?

We need to express:

• keys
• functional dependencies

Expressing keys is very easy, we use the PRIMARY KEY and UNIQUE keywords.

Expressing functional dependencies is possible also by means of a CHECK condition.

• What we need to say for the relation SkLoRo is that each tuple satisfies the following condition:

There are no tuples in the relation with the same value of Ro and different values of Lo.
CREATE TABLE SkLoRo
(Sk ..., Lo ..., Ro ..., UNIQUE (Sk,Ro), PRIMARY KEY (Sk,Lo), CHECK (NOT EXISTS SELECT *
FROM SkLoRo AS Copy
WHERE (SkLoRo.Ro = Copy.Ro
AND NOT SkLoRo.Lo = Copy.Lo)));

But this is generally not supported by actual relational database systems

Even assertions are frequently not supported
Can do it differently
Whenever there is an insert or update, check that FDs hold, or reject these actions
Maintaining FDs During Insertion

- We have a table \( R \) satisfying some FDs
- We have a table \( T \) of “candidates” for inserting into \( R \)
- We want to construct a subset of \( U \) of \( T \) consisting only of those tuples whose insertion into \( R \) would not violate FDs

- We show how to do it for the simple example of \( R = EGS \), where we need to maintain:
  - \( E \) is the primary key
  - \( G \to S \) holds

- We replace

\[
\text{INSERT INTO } R \\
(\text{SELECT } * \\
\text{FROM } T) ;
\]

By the following
Maintaining FDs During Insertion

INSERT INTO R
(SELECT *
FROM T
WHERE NOT EXISTS
(SELECT *
FROM R
WHERE (R.G = T.G AND R.S <> T.S) OR (R.E = T.E)
)
);

The WHERE condition will only insert only those tuples from T to R that satisfy the conditions:

- There is no tuple in R with the same value of the primary key E (assuming the system insists on this)
- There is no tuple in R with the same G but a different S
What If You Are Given A Decomposition?

- You are given a relation $R$ with a set of dependencies it satisfies
- You are given a possible decomposition of $R$ into $R_1, R_2, \ldots, R_m$
- You can check
  - Is the decomposition lossless: must have
  - Are the new relations in some normal forms: nice to have
  - Are dependencies preserved: nice to have
- Algorithms exist for all of these, which you could learn, if needed and wanted
- We do not have time to do it in this class
**Denormalization**

- After Normalization, we may want to **denormalize**
- The idea is to introduce redundant information in order to speed up some queries
- So the design not so clean, but more efficient
**DB Design Process**  
*(Roadmap)*

- Produce a good ER diagram, thinking of all the issues
- Specify all dependencies that you know about
- Produce relational implementation
- Normalize each table to whatever extent feasible
- Specify all assertions and checks
- Possibly denormalize for performance
  - May want to keep both EGS and GS
  - This can be done also by storing EG and GS and defining EGS as a view
A Review And Some Additional Material
What We Will Cover Here

- Review concepts dealing with Functional Dependencies
- Review algorithms
- Add some material extending previous material
Functional Dependencies
(Abbreviation: FDs)

Let X and Y be sets of columns, then:

X functionally determines Y, written $X \rightarrow Y$

if and only if

any two rows that are equal on (all the attributes in) X must be equal on (all the attributes in) Y

In simpler terms, less formally, but really the same, it means that:

If a value of X is specified, it “determines” some (specific) value of Y; in other words: Y is a function of X

We will assume that for a given FD $X \rightarrow Y$ attributes in X cannot have the value of NULL: attributes of X correspond to arguments of a function

We will generally look at sets of FDs and will denote them as needed by $M$ and $N$
Trivial FDs

If $Y \subseteq X$ then FD $X \rightarrow Y$

- Holds always
- Does not say anything

Such FD is called **trivial**

Can always remove the “trivial part” from an FD without changing the constraint expressed by that FD

Example: Replace

$$ABCD \rightarrow CDE$$

by

$$ABCD \rightarrow E$$

Having CD on the right side does not add anything
**Union Rule/Property**

- An FD with n attributes on the right hand side
  \[ X \rightarrow A_1 \ A_2 \ldots \ A_n \]
  is equivalent to the set of n FDs
  \[ X \rightarrow A_1 \]
  \[ X \rightarrow A_2 \]
  \[ \ldots \]
  \[ X \rightarrow A_n \]

- Example:
  \[ ABC \rightarrow DEFG \]
  is equivalent to set of 4 FDs
  \[ ABC \rightarrow D \]
  \[ ABC \rightarrow E \]
  \[ ABC \rightarrow F \]
  \[ ABC \rightarrow G \]
Closures of a Sets of Attributes

In general, we have a concept of a \textit{the closure} of a set of attributes in a relational schema $R$.

We are given a set of functional dependencies, say $M$.

Let $X$ be a set of attributes,

$X_M^+$ is the set of all the attributes whose values are "determined" by the values of $X$ because of $M$.

- If $M$ is understood, we do not need to write it and can just write $X^+$.
There is a very simple algorithm to compute $X^+$ (given some set of FDs):

1. Let $Y = X$
2. Whenever there is an FD, say $V \rightarrow W$, such that
   
   1. $V \subseteq Y$, and
   2. $W - Y$ is not empty
   
   add $W - Y$ to $Y$
3. At termination $Y = X^+$

The algorithm is very efficient

Each time we look at all the functional dependencies
- Either we can apply at least one functional dependency and make $Y$ bigger (the biggest it can be are all attributes), or
- We are finished
Given R (relation schema which is always denoted by its set of attributes), satisfying a set of FDs, a set of attributes X of R is a key (in information technology sometimes called “candidate key”), if and only if:

- \( X^+ = R \).
- For any \( Y \subseteq X \) such that \( Y \neq X \), we have \( Y^+ \neq R \).

Note that if R does not satisfy any (nontrivial) FDs, then R is the only key of R.

Example, if a table is R(FirstName,LastName) without any functional dependencies, then its key is just the pair (FirstName,LastName)
We are given $R$ (relation schema) and $M$ (set of FDs)

We have an anomaly whenever

$X \rightarrow Y$ is non-trivial and holds

but

$X$ does not contain a key of $R$ (sometimes phrased: $X$ is not a superkey of $R$)

Because there could be different tuples with the same value of $X$ and they all have to have the same value of $Y$

A relation is in BCNF if anomalies as described in this slide do not happen
How To Prove That A Relation Is Not In BCNF

To prove that relation $R$ is not in BCNF it is enough to show that there is a non-trivial FD $X \rightarrow Y$ and $X$ does not contain a key of $R$.

And to show that $X$ does not contain a key of $R$ it is enough to show that $X^+ \neq R$. 
Some Normal Forms

We have discussed several additional normal forms pertaining to FDs

- Second Normal Form (2NF)
- Third Normal Form (3NF)

We did not look at the most general definitions

Let us review what we did using an old example

We have, in general, FDs of the form \( X \rightarrow Y \)

But by the union rule, we can decompose them and consider FDs of the form \( X \rightarrow A \), where \( A \) is a single attribute
Classification Of FDs
(Our Old Example Focusing Only on One Key)

The three “not from the full key” dependencies are classified as:

- **Partial dependency**: From a part of the primary key to outside the key
- **Transitive dependency**: From outside the key to outside the key (this our informal phrasing)
- **Into key dependency**: From outside the key into (all or part of) the key

But what if we have $X \rightarrow Y$ where $X$ is partially in the key and partially outside the key?
It is Incomplete to Focus on Only One Key (The Primary Key)

By looking at the diagram we immediately can deduce that ST is also a key
- Because T determines C and therefore as SC determined R, so did ST

And we discussed it too.
General Definition of Some Normal Forms

Let R be relation schema

We will list what is permitted for three normal forms

We will include an obsolete normal form, which is still sometimes considered by practitioners: second normal form (2NF)

It is obsolete, because we can always find a desired decomposition in relations in 3NF, which is better than 2NF

The interesting one is a general definition of 3NF

Note: no discussion of which key is chosen to be primary as this is formally really “an arbitrary decision” though perhaps important for the application
### Which FDs Are Allowed For Some Normal Forms

Consider $X → A$ (X a set, A a single attribute)

<table>
<thead>
<tr>
<th>BCNF</th>
<th>3NF</th>
<th>2NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X → A$ is trivial (A is inside X)</td>
<td>$X → A$ is trivial (A is inside X)</td>
<td>$X → A$ is trivial (A is inside X)</td>
</tr>
<tr>
<td>X contains a key</td>
<td>X contains a key</td>
<td>X contains a key</td>
</tr>
<tr>
<td></td>
<td>A is in some key (informally: into a key, but X can overlap a key)</td>
<td>A is in some key (informally: into a key, but X can overlap a key)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X not a proper subset of any/some key</td>
</tr>
</tbody>
</table>
Cannot Have an FD From a Key Into Itself

- It is not possible to have a non-trivial functional dependency from a part of key into that same key.
- Proof by example:

In such a situation ABC is “too big” and actually BC is a key (and also the drawing does not follow standards).
Example: Relation in 3NF And Not in BCNF

Given functional dependencies: ABC → DE and CD → A

ABC is a key and designated as primary

This relation is not in BCNF as we have CD → A and CD does not contain a key as is easily seen

But CD → A is of the form: (something not containing a key) → (attribute in a key) and this is permitted by 3NF

Note there is another key that could have been the primary key: BCD

Originally people were confused as they considered only one key and did not realize that in general 3NF ≠ BCNF
If Only One Key Then 3NF \implies BCNF

Proof by contradiction (using example, but really general)
Assume that a relation is in 3NF but not in BCNF and there is only one key
Then we have a functional dependency that is permitted by 3NF but not permitted by BCNF, that is of the form
(something not containing a key) \rightarrow (attribute in a key)

Example

ABC is a key and CD \rightarrow A holds

Then we see that BCD is a key also, so we have more than one key
So we proved: if 3NF and only one key then BCNF
Relative Power of FDS: Simplify RHS

- If attributes removed from RHS (right hand side), the functional dependency becomes weaker.

- Changing from $\text{ABCD} \rightarrow \text{EFG}$ to $\text{ABCD} \rightarrow \text{EF}$ the dependency becomes weaker.

- Intuitively, after the simplification, we start with the same assumptions and deduce fewer conclusions.
Relative Power of FDS: Simplify LHS

- If attributes removed from LHS (left hand side), the functional dependency becomes stronger.

- Changing from $ABCD \rightarrow EFG$ to $ABC \rightarrow EFG$ the dependency becomes stronger.

- Intuitively, after the simplification, we start with fewer assumptions and deduce the same conclusions.
A Typical Step in Computing Minimal Cover

- We have a set $M$ of functional dependencies
- $M$ contains two functional dependencies with the same left hand side, say
  - $X \rightarrow EFG$
  - $X \rightarrow GH$
- We replace these functional dependencies by one functional dependency
  - $X \rightarrow EFGH$
- And we get a set $N$ of functional dependencies

- $N$ is equivalent to $M$
A Typical Step in Computing Minimal Cover

- We have a set $M$ of functional dependencies.
- $M$ contains a functional dependency with more than one attribute in the RHS, say $X \rightarrow EFG$
- We replace this functional dependency by $X \rightarrow EF$
- And we get a set $N$ of functional dependencies

- $N$ can only be weaker (in power) than $M$
- $N$ is equivalent (in power) to $M$ if and only if we can “prove the stronger functional dependency”: $X_N^+ \text{ contains EFG}$. 

© 2014 Zvi M. Kedem
A Typical Step in Computing Minimal Cover

We have a set M of functional dependencies.

M contains a functional dependency with more than one attribute in the LHS, say

\[ AB \rightarrow C \]

We replace this functional dependency by

\[ AC \rightarrow B \]

And we get a set N of functional dependencies

N can only be stronger (in power) than M

N is equivalent to M

if and only if

we can “prove the stronger functional dependency”:

\[ ABCM^+ \text{ contains } Y \]
The Goal

Given a table $R$ satisfying a set of FDs $M$, decompose it into tables: $R_1$ satisfying $M_1$, $R_2$ satisfying $M_2$, ..., $R_k$ satisfying $M_k$, such that

- The decomposition is lossless join: can recover $R$ from $R_1$, $R_2$, ..., $R_k$ using natural join
- Dependencies are preserved: making sure that (after changes to the database) if $R_1$ satisfies $M_1$, $R_2$ satisfies $M_2$, ..., $R_k$ satisfies $M_k$, then that if we recover $R$ it will satisfy $M$
- $R_1$, $R_2$, ..., $R_k$ are all in 3NF (and if we are lucky also in BCNF)
Sketch of The Procedure

- Compute a minimal cover $N$ for $M$
- Create a table for each functional dependency in $N$
- Remove redundant tables that is
  - Remove a table if its set of columns is a subset of the set of columns of another table
- Check if at least one table contains a global key: just compute closure of its attributes using $M$ (or $N$, likely faster) and see if you get all of $R$
- If no table contains a global key, find one global key (using heuristics or otherwise) and add a table whose columns are the attributes of the global key you found
Key Ideas

- Need for decomposition of tables
- Functional dependencies
- Some types of functional dependencies:
  - Partial dependencies
  - Transitive dependencies
  - Into full key dependencies
- First Normal Form: 1NF
- Second Normal Form: 2NF
- Third Normal Form: 3NF
- Boyce-Codd Normal Form: BCNF
- Removing redundancies
- Lossless join decomposition
- Preservation of dependencies
- 3NF vs. BCNF
Key Ideas

- Multivalued dependencies
- Fourth Normal Form: 4NF
- Minimal cover for a set of functional dependencies
- Algorithmic techniques for finding keys
- Algorithmic techniques for computing a minimal cover
- Algorithmic techniques for obtaining a decomposition of relation into a set of relations, such that
  - The decomposition is lossless join
  - Dependencies are preserved
  - Each resulting relation is in 3NF
- Denormalization after Normalization