Armstrong's Deductive System for FDs:

From	We can conclude	Rule #
	$\mathbf{AX \to X}$	1
	$\mathbf{A \to A}$	2
$\mathbf{A \to B}$	$\mathbf{AC \to BC}$	3
$\mathbf{A \to B}$ and $\mathbf{B \to C}$	$\mathbf{A \to C}$	4
$\mathbf{A \to BC}$	$\mathbf{A \to B}$ and $\mathbf{A \to C}$	5
$\mathbf{A \to B}$	$\mathbf{AC \to B}$	6
$\mathbf{A \to B}$ and $\mathbf{A \to C}$	$\mathbf{A \to BC}$	7
$\mathbf{A \to B}$ and $\mathbf{C \to D}$	$\mathbf{AC \to BD}$	8

Two Axioms plus Deduction Rules

- $\mathbf{A}, \mathbf{X}, \mathbf{B}, \ldots$ are (sets of) attributes (for the same relation schema)
- The first two can be seen as logically valid formulas

  The other six, as logical deduction rules
- The axioms and rules in red (1,3,4) are good enough for our purposes

  The other axioms and rules are logically redundant, i.e. they can be derived from the three main ones

  But it is useful to keep and use them

- Officially, Armstrong's rules for deriving FDs are the following:

$$A1: \frac{}{\mathbf{AX} \to \mathbf{X}} \qquad A2: \frac{\mathbf{A} \to \mathbf{B}}{\mathbf{AC} \to \mathbf{BC}} \qquad A3: \frac{\mathbf{A} \to \mathbf{B},\ \mathbf{B} \to \mathbf{C}}{\mathbf{A} \to \mathbf{C}}$$

- They are read and used as follows:

  If at some point of a derivation of a FD we have (an instantiation of) the upper part, then we can jump to the (corresponding instantiation of the) bottom part

- In particular, *A1* can be applied without any "hypothesis"
- These are the inclusion, augmentation, transitivity rules, resp.
- To derive implicit FDs, use the deductive system as follows:
  1. Start from a given set of FDs $\mathcal{F}$

     Usually those explicitly stated with a relation schema

  2. Apply iteratively (and mechanically, symbolically) the rules above, with the original and already derived FDs

     Deriving new FDs ψ until no new FD is obtained

- If the FD ψ can be derived from a set $\mathcal{F}$ of FDs using Armstrong's system, we use the notation: $\mathcal{F} \vdash_A \psi$
- Example: Derive Rule 5 using the three official rules

  Hypothesis: $\mathbf{A} \to \mathbf{BC}$ $\quad (*)$

  To deduce: $\mathbf{A} \to \mathbf{B}$ and $\mathbf{A} \to \mathbf{C}$

  Using Rule *A1*, introduce without any hypothesis

  $$\mathbf{BC} \to \mathbf{B} \qquad (**)$$

  Using rule *A3* combining $(*)$ and $(**)$, obtain: $\mathbf{A} \to \mathbf{B}$

  The other part is similar
- Here we did the formal derivation:

  1. $\mathbf{A} \to \mathbf{BC}$ (hypothesis, the only element of $\mathcal{F}$ in this case)
  2. $\mathbf{BC} \to \mathbf{B}$ (by rule *A1*)
  3. $\mathbf{A} \to \mathbf{B}$ (by rule *A3*; the ψ above)

  Something a computer algorithm could do, symbolically ...

- Exercise: Obtain all the other redundant rules using the three official rules (and previously derived ones)
- Example: Relation schema

  $EmpInfo(Id, Name, Phone, Dept, SkillId, SkillName, SkillDate, SkillLvl)$

  Assume following FDs hold:

  $$Id \rightarrow Name, \quad Id \rightarrow Phone, \quad Id \rightarrow Dept \qquad (*)$$

  Using the Union Rule (# 7), obtain:

  $$Id \rightarrow Name\ Phone\ Dept \qquad (**)$$

  We can use derived rules, because they can be justified by (derived from) the three official rules

  Then:

  $\{Id \rightarrow Name,\ Id \rightarrow Phone,\ Id \rightarrow Dept\} \vdash_A Id \rightarrow Name\ Phone\ Dept$

  Notice: from $(**)$ one can derive each of the FDs in $(*)$ using the decomposition rule (# 5)

- How good is the process of deriving FDs using Armstrong's deductive system (ADS)?

  More precisely?

- Can every FD ψ that is implied by a set of FDs $\mathcal{F}$ be deduced using ADS?

  Is it strong enough?

- Conversely, is every FD ψ obtained from $\mathcal{F}$ via ADS correct?

  In the sense that it is implied by $\mathcal{F}$?

- Yes to both! This is Armstrong's Theorem (that can be mathematically proved)

  With ADS one can derive all and only the FDs that are implied by the original set $\mathcal{F}$

  In other terms: $\underbrace{\mathcal{F} \models \psi}_{\text{semantic notion}} \iff \underbrace{\mathcal{F} \vdash_A \psi}_{\text{syntactic, symbolic notion}}$

- $$\underbrace{\mathcal{F} \models \psi}_{\text{semantic notion}} \iff \underbrace{\mathcal{F} \vdash_A \psi}_{\text{syntactic notion}}$$
- The LHS is semantic level (metalevel) in that it appeals to all possible instances of a relation schema

  This is about implications we -humans- establish via mathematical proofs
- The RHS is symbolic (object level) in that it is about symbolic manipulation of symbolic expressions

  This is about derivations that humans and computer programs (for symbolic processes) can do
- "$\Longleftarrow$" is easy to prove (showing we always produce correct FDs, as on page 61)

  "$\Longrightarrow$" is more complicated
- We can delegate to a computer (program) the derivation of implicit FDs

Other Computational Problems around FDs:

- Several and important, not all solvable directly via ADS

1. Compute all the FDs that are entailed by a given set of FDs?

   The deductive closure $\mathcal{F}^+$ of a set $\mathcal{F}$ of FDs

   $\mathcal{F}^+$ can be computed in polynomial time in the size of $\mathcal{F}$

   (Beeri and Bernstein, ACM TODS, 1979, 4(1): 30-59)

   A fixed relation schema has a fixed and finite set of attributes; and potential FDs are built considering combinations of attributes (for the LHS and RHS of the implication), and there are finitely many possible combinations Of course, not a guarantee of efficiency

2. The decision problem: $\mathcal{F} \stackrel{?}{\models} \psi$

   For a fixed set $\mathcal{F}$ of FDs (and schema), it is about deciding membership of the problem $\{\varphi \mid \varphi \text{ is an FD, and } \mathcal{F} \models \varphi\}$

   It is solvable in polynomial time

   Compute $\mathcal{F}^+$ and check if $\psi \in \mathcal{F}^+$

3. Computing a minimal cover for a set of FDs

   That is, a minimal set of FDs that imply the given set $\mathcal{F}$

   More precisely, a set $\mathcal{G}$ of FDs such that:

   - $\mathcal{G}^+ = \mathcal{F}^+$, and
   - For no $\mathcal{G}' \subsetneq \mathcal{G}$: $\mathcal{G}'^+ = \mathcal{F}^+$

   This can be done efficiently

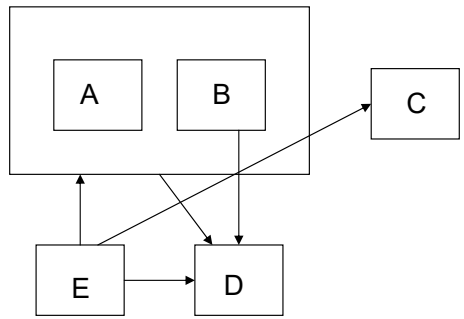
   Useful for normalization and checking violations more efficiently

Normalization: Normal Forms

- Diagrams of Dependencies for attributes and their FDs

  For a relation schema

  $R(A, B, C, D, E, \ldots)$



A B → D
E → A B
E → C
B → D
E → D

- Notions associated to keys (based on FDs):
  - Superkey: Set of attributes that functionally determine all the other attributes (in the relation)
  - Candidate Key: (simply, a key)

    Minimal Superkey: No proper subset is also a superkey
  - Primary Key: One of the candidate keys chosen as primary
  - Foreign Key: Set of attributes of a relation R that is a primary key for a relation S

Update Anomalies: (w.r.t. changes of attribute values)

- As already mentioned, considerations around them become relevant for normalization
- They commonly appear in the presence (or caused) by data redundancy
- Intuitively, they happen on a relation if changing an attribute value forces to update several tuples in the table
- Example:

Employee	ID	Name	PhoneExt	Skill	Level
	111	john	1778	paint	good
	111	john	1778	carpenter	medium
	111	john	1778	carpenter	medium
	145	peter	1750	carpenter	good

  If we change the phone extension in 1st row, it has to be changed in the other two (if we want $ID \rightarrow PhoneExt$)
- This may lead to inconsistencies, administrative overhead, ...
- Better eliminate attribute *PhoneExt* from the table?

  Creating a separate one with IDs and phone extensions

Deletion Anomalies:

- A relation is subject to them when deleting a tuple to reflect the disappearance of an entity (element of an entity set) or relationship may cause losing information about some other entity of a different entity set or a relationship
- It is useful to think a tuple (a row) in a table as describing an individual entity; or several if they are in relationship

  Here we are using "entity" for "element of an ER entity"
- Example:

  Drinks(Wine#, Vineyard, Quality, Year, Drinker#, DrinkerName, Address)

  If we eliminate all the information about drinkers, we lose information about wines, e.g. about vineyards

  (unless we fill the table with null values)
- Better keep info about wines and drinkers in separate tables

  Create a 3rd table with codes *Wine#*, *Drinker#* to represent the relationship

  Plus FKCs

Insertion Anomalies:

- Similar to deletions anomalies, but for insertions
- Not possible (or undesirable) to independently insert values for a single entity
- We cannot represent information about some entity without including information about some other entity or relationship that does not exist
- *Drinks(Wine#, Vineyard, Quality, Year, Drinker#, DrinkerName, Address)*

  If we want to insert info about wines only, what about the drinkers?

  This will cause the insertion of many null values

Back to Normalization:

- Normalization is a sequence of decompositions of "wide" relation schema (possibly, a universal relation schema) into new relation schemas

  The resulting relational schema does not present the above mentioned anomalies

- Normalization conditions are checked on a relation schema

  With its associated dependencies

- A relation schema is normalized, i.e. decomposed into other relation schemas

  With newly generated dependencies

- The new relational schema satisfies the normalization conditions (to be defined next)

- Normalization conditions are not checked or imposed on instances

Example: (to be used throughout the rest of the chapter)

- Start with a wide relation schema with attributes:

```
#emp,    emp_name,    emp_phone

dep_name,    dep_phone,    dep_man

#skill,   skill_name,  skill_date,  skill_level
```

- Assumptions:

- `emp` attributes refer to data about (entity) employees
  They are uniquely identified by `#emp`
- `dep` attributes refer to (entity) departments
  `dep_name` uniquely identify departments
- `skill` items refer to abilities of employees
- Date refers to the last time an employee's ability was tested

```
#emp,    emp_name,    emp_phone

dep_name,    dep_phone,    dep_man

#skill,   skill_name,  skill_date,  skill_level
```

- The following FDs are assumed to hold:

  $$\texttt{\#emp} \rightarrow \texttt{emp\_name},\ \texttt{emp\_phone}, \texttt{dep\_name}$$
  $$\texttt{dep\_name} \rightarrow \texttt{dep\_phone},\ \texttt{dep\_man}$$
  $$\texttt{\#skill} \rightarrow \texttt{skill\_name}$$
  $$\texttt{\#emp},\ \texttt{\#skill} \rightarrow \texttt{skill\_date},\ \texttt{skill\_level}$$

- From these we can see (and deduce):

  {#emp, #skill} is a key for the relation schema

  (verify this claim!)

First Normal Form (1NF)

- A basic normalization condition to start with

  Usually implicitly required

  As opposed to those that follow, it is not based on FDs

- A relation is in 1NF if all its attributes are single-valued

  That is, attributes take atomic values, not set-values

- A condition imposed on the schema, as all the normalization conditions

  It restricts the data domains for the attributes, and instances of the relation schema

- A relation with attributes that are not single-valued can be replaced by one that does have the required property

Example:

#emp	dept.	degrees
e234	Hydraulics	{Engineer, Master, PhD}
e341	Systems	{Engineer, MBA}
...	...	...

#emp	dept.	degree
e234	Hydraulics	Engineer
e234	Hydraulics	Master
e234	Hydraulics	PhD
e341	Systems	Engineer
e341	Systems	MBA
...	...	...

- Not necessarily "good", but 1NF
- In general, RDBMSs do not allow the creation (population) of relations that are not in 1NF
- They are difficult to update (update elements of sets?)

  Or check for FD satisfaction, among other issues

A universal relation in 1NF for the running example:

emp_info

#emp	emp_name	· · ·	#skill	skill_name	skill_date	skill_level
09112	Jara	· · ·	44	librarian	Mar/95	12
09112	Jara	· · ·	26	mecanog	Jun/97	10
09112	Jara	· · ·	89	word-proc	Ene/98	12
12231	Soto	· · ·	26	mecanog	Abr/97	5
12231	Soto	· · ·	39	archivist	Jul/97	7
13597	Brown	· · ·	26	mecanog	Sep/97	6
14131	Barros	· · ·	26	mecanog	May/97	9
14131	Barros	· · ·	89	word-proc	Sep/97	10
· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·

- FDs: $\texttt{\#emp} \rightarrow \texttt{emp\_name}, \texttt{emp\_phone}, \texttt{dep\_name}$

  $\texttt{dep\_name} \rightarrow \texttt{dep\_phone}, \texttt{dep\_man}$

  $\texttt{\#skill} \rightarrow \texttt{skill\_name}$

  $\texttt{\#emp}, \texttt{\#skill} \rightarrow \texttt{skill\_date}, \texttt{skill\_level}$

  A candidate key is $\{\texttt{\#emp}, \texttt{\#skill}\}$
- All FDs are satisfied by relation (instance) emp_info above
- Table presents several anomalies

  Revealing a bad design
- 1NF is not enough

emp_info

#emp	emp_name	· · ·	#skill	skill_name	skill_date	skill_level
09112	Jara	· · ·	44	librarian	Mar/95	12
09112	Jara	· · ·	26	mecanog	Jun/97	10
09112	Jara	· · ·	89	word-proc	Ene/98	12
12231	Soto	· · ·	26	mecanog	Abr/97	5
12231	Soto	· · ·	39	archivist	Jul/97	7
13597	Brown	· · ·	26	mecanog	Sep/97	6
14131	Barros	· · ·	26	mecanog	May/97	9
14131	Barros	· · ·	89	word-proc	Sep/97	10
· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·

$$\texttt{\#emp} \rightarrow \texttt{emp\_name},\ \texttt{emp\_phone}, \texttt{dep\_name}$$

$$\texttt{dep\_name} \rightarrow \texttt{dep\_phone},\ \texttt{dep\_man}$$

$$\texttt{\#skill} \rightarrow \texttt{skill\_name}$$

$$\texttt{\#emp},\ \texttt{\#skill} \rightarrow \texttt{skill\_date},\ \texttt{skill\_level}$$

Candidate key: $\{\texttt{\#emp},\ \texttt{\#skill}\}$

- Every time a `#skill` value appears, the same `skill_name` has to appear
- Every time we want to add a skill to an employee, we have to repeat all the employee's info, e.g. `emp_name`
- Notice that both `skill_name` and `emp_name` functionally depend on a part of the key

  And they do not belong to any candidate key

Second Normal Form (2NF)

- A relation schema is in 2NF if it is in 1NF, and

  Every attribute not belonging to a candidate key does not depend only on a proper subpart of a candidate key

- More formally:

  For every candidate key **A** and attribute `A` not belonging to any candidate key (in particular, `A` $\notin$ **A**), there is no **A'** $\subsetneq$ **A**, such that **A'** $\rightarrow$ `A`

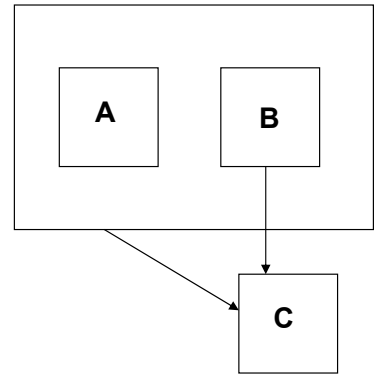
- Equivalently: **A** $\rightarrow$ `A`, and `A` $\notin$ **A**, and `A` does not belong to any candidate key $\implies$ **A** is not proper subset of any key

- Relation (schema) `emp_info` on page 83 is not in 2NF

  E.g. `skill_name` depends only upon `#skill`, etc.

- A relational schema is in 2NF if each of its relation schemas is in 2NF (w.r.t. their FDs)

Example: Relation schema $R(\mathtt{A}, \mathtt{B}, \mathtt{C})$, with dependencies as shown



$\mathtt{A}$ is #student
$\mathtt{B}$ is course_section
$\mathtt{C}$ is instructor_name

- It is not in 2NF:

  $\mathtt{\#student}, \mathtt{course\_section} \rightarrow \mathtt{instructor\_name}$

  $\mathtt{course\_section} \rightarrow \mathtt{instructor\_name}$

  - Neither $\mathtt{A} \not\rightarrow \mathtt{B}$ nor $\mathtt{B} \not\rightarrow \mathtt{A}$: $\{\mathtt{A}, \mathtt{B}\}$ is a candidate key
  - $\mathtt{C}$ depends only partially upon the key $\{\mathtt{A},\mathtt{B}\}$

    On a proper subset of the key (on $\{\mathtt{B}\}$)
  - $\mathtt{C}$ does not belong to any candidate key:

    If it is added to the only (candidate) key, $\{\mathtt{A},\mathtt{B}\}$, the result is non-minimal ($\mathtt{C}$ can be discarded)
- What consequences?

- There are some problems if we have those attributes in a single relation course(A,B,C)
  - We cannot add info about a course and its instructor without having students
  - For each student in the same section, the same instructor name has to be repeated
- The normalization process is based on the decomposition of relation schemas into relation schemas with a smaller number of attributes

  To eliminate anomalies like those above
- Not every decomposition performed to achieve 2NF has desirable properties
- What are good properties of a decomposition?
- We need to investigate properties decompositions in their own right

Decompositions:

$T(A, B, C)$

A	B	C
a1	100	c1
a2	200	c2
a3	300	c3
a4	200	c2

$T_1(A, B)$

A	B
a1	100
a2	200
a3	300
a4	200

and

$T_2(B, C)$

B	C
100	c1
200	c2
300	c3

- Some natural requirements are satisfied here
  - Set of attributes of T is the union of the sets of attributes of T_1 and T_2 (allowing shared attributes)
  - For every DB instance D, the instances $T_1(D), T_2(D)$ are obtained from $T(D)$ by projection on the corresponding attributes

    (So, T_1, T_2 can be seen as projection views defined on T)
- Are these the only good properties we expect from a decomposition?

- Through the decomposition process, we also expect to preserve information

  Both positive and negative: no data added, no data lost

- Can we always put back together the original information by joining the decomposed data?

- This should depend only on the participating relation schemas and dependencies

  For all possible instances!

- What conditions on the participating schema and subschemas and the dependencies thereof to ensure no information loss?

$T(A, B, C)$

A	B	C
a1	100	c1
a2	200	c2
a3	300	c3
a4	200	c4

$\rightsquigarrow$

$T_1(A, B)$

A	B
a1	100
a2	200
a3	300
a4	200

and

$T_2(B, C)$

B	C
100	c1
200	c2
300	c3
200	c4

$T(D) \subsetneq T_1(D) \bowtie_B T_2(D)$

(negative information about $T(D)$ lost)

- Decomposition of a relation schema T into relation schemas $T_1, \ldots, T_k$ With $Attr(T) = \bigcup_i Attr(T_i)$
- For $T(D)$ an instance for T: $T_i(D) := \Pi_{Attr(T_i)} T(D)$

  $T_i(D)$ becomes an instance for T_i
- The decomposition is lossless if for every instance $T(D)$ of T:

$$T(D) = T_1(D) \bowtie T_2(D) \bowtie \ldots \bowtie T_k(D) \qquad (*)$$

- Implicit requirement: the smaller schemas share attributes for the join; otherwise, use the cartesian product
- The decomposition on page 88 is not lossless

  One instance is good enough as a counterexample
- How can we check that a decomposition is lossless?
- Appealing to all possible instances?
- Not very practical or possible ...

- Anything better? More operational? Based on what?
- Assume we have instead:

  With the FD $B \rightarrow C$ on the schema

  Satisfied by this instance
  (but not by the one on page 88)

$T(A, B, C)$

A	B	C
a1	100	c1
a2	200	c2
a3	300	c3

$T(A, B, C)$

A	B	C
a1	100	c1
a2	200	c2
a3	300	c3
a4	200	c2

$\rightsquigarrow$

$T_1(A, B)$

A	B
a1	100
a2	200
a3	300
a4	200

and

$T_2(B, C)$

B	C
100	c1
200	c2
300	c3

Now: $T(D) = T_1(D) \bowtie_B T_2(D)$

- Problematic combination was related to the lack of this FD
- Let us try to uncover conditions based on the presence of (satisfied) FDs that guarantee a lossless decomposition