**Containment of a FD in a closure $F^+$**

- question: Let $F$ be a set of FDs and $A \rightarrow B$ a FD. Does $A \rightarrow B \in F^+$ hold?
- problem: explicit calculation of $F^+$ is too expensive
- instead: calculation of the closure $A^+$ of the attribute set $A$ regarding the set $F$
  - $A^+$ consists of all attributes that are functionally determined by $A$.
  - If $B \subseteq A^+$ holds, then also $A \rightarrow B \in F^+$ holds.

- algorithm for inferring $A^+$

  **algorithm** AttrClosure($F$, $A$)
  
  // input: a set $F$ of FDs and a set $A$ of attributes
  // output: the complete set $A^+$ of attributes for which holds: $A \rightarrow A^+$

  $A^+ := A$;
  repeat
      Old$A^+$ = $A^+$;
      foreach FD $B \rightarrow C \in F$ do
          if $B \subseteq A^+$ then $A^+ := A^+ \cup C$;
      until $A^+ = OldA^+$; return $A^+$
Canonical cover

- In general, distinct equivalent sets of FDs exist. Two sets $F$ and $G$ of FDs are called **equivalent** iff $F^+ = G^+$ holds.

- Definition of equivalence is convincing, because the equality of the closures for $F$ and $G$ implies that the same FDs can be inferred from $F$ and $G$.

- Conclusion: For a given set $F$ of FDs there exists a unique closure $F^+$.

- Drawbacks of the closure $F^+$:
  - in general very many FDs in $F^+$ so that the handling with $F^+$ becomes difficult
  - large redundant set of FDs that has to be checked as consistency tests for database modifications

- Goal: computation of a most possible small set of FDs which are equivalent to $F$
  → less effort for testing whether a new or updated tuple violates a FD
- $F_c$ is called **canonical cover** of a given set $F$ of FDs, if holds:
  
  - $F_c^+ = F^+$
  
  - In $F_c$ there are no FDs $A \rightarrow B$ where $A$ or $B$ contain *extraneous* attributes, i.e.,
    
    $$\forall a \in A : (F_c - \{A \rightarrow B\} \cup \{(A - \{a\}) \rightarrow B\})^+ \neq F_c^+$$
    
    $$\forall b \in B : (F_c - \{A \rightarrow B\} \cup \{A \rightarrow (B - \{b\})\})^+ \neq F_c^+$$
  
  - Each left side of the FDs in $F_c$ occurs only once, i.e.,
    
    if $A \rightarrow B$ and $A \rightarrow C$ hold, then in $F_c$ only the FD $A \rightarrow B \cup C$ is used.

- **algorithm for computing the canonical cover**
  
  - **step 1**: For each FD $A \rightarrow B \in F$ perform the **left reduction**: check for all $a \in A$ whether the attribute $a$ is extraneous, i.e., whether
    
    $$B \subseteq \text{AttrClosure}(F, A - \{a\})$$
    
    holds. If this is the case, replace $A \rightarrow B$ by $(A - \{a\}) \rightarrow B$. 
− step 2: For each remaining FD $A \rightarrow B \in F$ perform the **right reduction**: check for all $b \in B$, whether the attribute $b$ is extraneous, i.e., whether

$$b \in AttrClosure(F - \{A \rightarrow B\} \cup \{A \rightarrow (B - \{b\})\}, A)$$

holds. If this is the case, replace $A \rightarrow B$ by $A \rightarrow (B - \{b\})$.

− step 3: Remove the FDs of the form $A \rightarrow \emptyset$ which perhaps have been produced in the previous step.

− step 4: By using the union rule replace all FDs of the form $A \rightarrow B_1, \ldots, A \rightarrow B_n$ by $A \rightarrow B_1 \cup \ldots \cup B_n$

![example](image)

− Given the set $F = \{A \rightarrow B, B \rightarrow C, A \cup B \rightarrow C\}$.

− step 1: $A \cup B \rightarrow C$ is replaced by $A \rightarrow C$, because $B$ on the left side is extraneous ($C$ is already functionally dependent from $A$ by the first two FDs).

− step 2: $A \rightarrow C$ is replaced by $A \rightarrow \emptyset$, because $C$ on the right side is extraneous. This results from the fact that $C \subseteq AttrClosure(\{A \rightarrow B, B \rightarrow C, A \rightarrow \emptyset\}, A)$.

− step 3: $A \rightarrow \emptyset$ is removed. We obtain: $F_c = \{A \rightarrow B, B \rightarrow C\}$.

− step 4: Nothing to be done.
Decomposition of a relation schema

- **Normalization**: In order to eliminate anomalies (redundancies, update, insertion and deletion anomalies), the relation schema $R$ is decomposed into $n$ relation schemas $R_1, \ldots, R_n$.

- Two fundamental correctness criteria for such a decomposition:
  - **Losslessness (lossless join decomposition)**: An arbitrary instance $r(R)$ must be reconstructable from the instances $r_1(R_1), \ldots, r_n(R_n)$.
  - **Dependency preservation**: All FDs which hold for schema $R$ must be transferable to the schemas $R_1, \ldots, R_n$ and must be efficiently checkable.

- **Losslessness**
  - It is sufficient to confine oneself to the decomposition of $R$ into two relation schemas $R_1$ and $R_2$.
  - Of course, we must require: $R = R_1 \cup R_2$.
  - A decomposition of $R$ into $R_1$ and $R_2$ is **lossless** if for all relations $r(R)$ holds:
    \[ r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r). \]
  - That is, reconstruction must be possible by natural join.
criteria for the losslessness of a decomposition

Let $R$ be a relation schema and $F_R$ the set of FDs. A decomposition of $R$ in $R_1$ and $R_2$ is lossless, if

$$(R_1 \cap R_2) \rightarrow R_1 \in F_R^+ \quad \text{or} \quad (R_1 \cap R_2) \rightarrow R_2 \in F_R^+$$

i.e., $R_1 \cap R_2$ is a **superkey** for $R_1$ or $R_2$.

Alternative formulation: Let $R = A \cup B \cup C$, $R_1 = A \cup B$ and $R_2 = A \cup C$ with pair-wise disjoint attribute sets $A, B, C$. Then:

$$B \subseteq \text{AttrClosure}(F_R, A) \quad \text{or} \quad C \subseteq \text{AttrClosure}(F_R, A)$$

must hold.

sufficient, but not necessary condition for losslessness

In a relation schema $R$, $X \subseteq R$ is called a **superkey**, if $X \rightarrow R$ holds.

example for a lossy decomposition:

The decomposition of the relation $R$(sname, saddr, product, price) in the two relations \textit{supplier}(sname, saddr, product) and \textit{offer}(product, price) is \textit{not} lossless, since in general $R \neq \textit{supplier} \bowtie \textit{offer}$ holds.

reasons:

+ Product does not functionally determine the price.
+ Product does not functionally determine supplier’s name and address.
dependency preservation

- goal: All FDs that hold for schema $R$ are to be checkable locally on each of the decomposed schemas $R_1, \ldots, R_n$ without the computation of joins (efficiency!).

- For that purpose determine for each $R_i$ the restriction $F_{R_i}$ of FDs in $F_R^+$, i.e., $F_{R_i}$ contains those dependencies of the closure of $F_R$ that contain only attributes of $R_i$. We require:

$$F_R^+ = (F_{R_1} \cup F_{R_2} \cup \ldots \cup F_{R_n})^+$$

(dependency-preserving decomposition)

example for a lossless but not dependency-preserving decomposition:

- given: schema $address$(street, city, state, zipcode).

- We assume the following simplified conditions:
  
  + Cities are uniquely characterized by their name (city) and their state (state).
  + Within a street the zipcode does not change.
  + Zipcode areas do not extend over city borders, and cities do not extend over state borders.

- FDs therefore: $\{\text{zipcode}\} \rightarrow \{\text{city, state}\}$, $\{\text{street, city, state}\} \rightarrow \{\text{zipcode}\}$

- Consider the decomposition of $address$ in $streets$(zipcode, street) and in $cities$(zipcode, city, state).
This decomposition is lossless, since zipcode is the only common attribute and \{zipcode\} → cities holds.

Since the FD \{street, city, state\} → \{zipcode\} cannot be assigned to one of the relations streets or cities, this decomposition is not dependency-preserving.

Normal forms

- By using FDs, we can define several normal forms that represent “good” database designs.

- assumptions for normalization:
  - A set of FDs is given for each relation.
  - Each relation has a primary key.

- This information combined with the conditions (constraints) for the different normal forms effects the normalization process.

- Some more general definitions of these normal forms consider all candidate keys instead of only the primary key.

- Further normal forms rest on other kinds of data dependencies.

- “relational design by means of analysis“
8.3 First Normal Form

- A relation schema is in **first normal form (1NF)**, if, and only if, the domains of all attributes contain only atomic values that cannot be subdivided any more.
- This property is a *fundamental component* of the relational model and is hence presupposed for further considerations.
- In particular: Composite, set-valued or even relation-valued attribute domains are not permitted.
- **NF²-relations (NF² = Non First Normal Form; nested relations)**
  - reason for introduction: The 1NF is frequently too inflexible when modeling data.
  - example:

<table>
<thead>
<tr>
<th></th>
<th>parents</th>
<th>father</th>
<th>mother</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ben</td>
<td>Martha</td>
<td>{Liza, Lucia}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ben</td>
<td>Maria</td>
<td>{Theo, Josef}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>John</td>
<td>Martha</td>
<td>{Cleo}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>father</th>
<th>mother</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ben</td>
<td>Martha</td>
<td>Liza</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lucia</td>
</tr>
<tr>
<td></td>
<td>Ben</td>
<td>Maria</td>
<td>Theo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Josef</td>
</tr>
<tr>
<td></td>
<td>John</td>
<td>Martha</td>
<td>Cleo</td>
</tr>
</tbody>
</table>
8.4 Second Normal Form

- A relation schema is in **second normal form (2NF)**, if, and only if, it is in 1NF and for all FDs $X \rightarrow \{A\}$ holds: If attribute $A$ is not part of a key and $X$ is a key, then there is no FD $Y \rightarrow \{A\}$ with $Y \subset X$.

- Alternative formulation:

  A relation schema is in **second normal form (2NF)**, if, and only if, it is in 1NF and each non-key attribute $A \in R$ is fully functionally dependent on each key $X$ of the schema, i.e., the FD $X \rightarrow \{A\}$ must hold, and this FD is left reduced.

- But: It is still possible for a relation in 2NF to exhibit transitive dependency; that is, one or more attributes may be functionally dependent on nonkey attributes.

- Example:
  - relation $StudentsLecture(reg-id, id, name, sem)$
  - corresponds to the join of the relations $attends$ and $students$
  - key $\{reg-id, id\}$ with all FDs having this key on the left side
    - in particular: $\{reg-id, id\} \rightarrow \{name\}$ and $\{reg-id, id\} \rightarrow \{sem\}$
  - additional FDs: $\{reg-id\} \rightarrow \{name\}$ and $\{reg-id\} \rightarrow \{sem\}$
  - $\Rightarrow$ violation of the 2NF
The following anomalies can occur:

+ insertion anomaly: What do we do with students who do not attend a lecture?
+ update anomaly: If a student reaches the next semester, we must ensure that in all tuples containing information about the student the semester number is changed accordingly.
+ deletion anomaly: What happens if a student drops his/her only lecture?

Solution of these problems is relatively simple: decompose the relation in several subrelations which each fulfil the 2NF. Split $StudentsLecture$ in the two relations $attend$(reg-id, id) and $students$(reg-id, name, sem). Both relations satisfy the 2NF. Moreover, they represent a lossless decomposition.

Remarks:

- no description of a decomposition algorithm which splits a given relation schema $R$ into several 2NF relation schemas $R_1$, ..., $R_n$, because always 3NF is strived for (low importance of 2NF)
- violation of 2NF only with composite keys
- conclusion: elimination of partial FDs between key and non-key attribute through 2NF