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:

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

(example of 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: \{zipcode\} $\rightarrow$ \{city, state\}, \{street, city, state\} $\rightarrow$ \{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”
7.3 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>father</th>
<th>mother</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td>Martha</td>
<td>{Liza, Lucia}</td>
</tr>
<tr>
<td>Ben</td>
<td>Maria</td>
<td>{Theo, Josef}</td>
</tr>
<tr>
<td>John</td>
<td>Martha</td>
<td>{Cleo}</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>father</th>
<th>mother</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cName</td>
</tr>
<tr>
<td>Ben</td>
<td>Martha</td>
<td>Liza</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lucia</td>
</tr>
<tr>
<td>Ben</td>
<td>Maria</td>
<td>Theo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Josef</td>
</tr>
<tr>
<td>John</td>
<td>Martha</td>
<td>Cleo</td>
</tr>
</tbody>
</table>
```
7.4 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 (i.e., full).

- 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 non-key attributes (example: relation $lecture(id, title, pers-id, room)$).

- 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 \( \text{StudentsLecture} \) in the two relations \( \text{attend}(\text{reg-id}, \text{id}) \) and \( \text{students}(\text{reg-id}, \text{name}, \text{sem}) \). 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 \( R_1, \ldots, R_n \) here, because always 3NF is the goal (low importance of 2NF)
- violation of 2NF only with composite keys
- conclusion: The 2NF eliminates the partial FDs between key and non-key attributes
7.5 Third Normal Form

Definition

A relation schema \( R \) with associated FDs \( F \) is in **third normal form (3NF)**, if, and only if, it is in 2NF and for each FD \( A \rightarrow B \in F \) at least one of the following conditions holds:

- \( B \subseteq A \), i.e., the FD \( A \rightarrow B \) is trivial.
- \( A \) is superkey of \( R \).
- \( B \) is (part of) some candidate key of \( R \).

These conditions exclude non-trivial FDs between non-key attributes. That is, transitive dependencies of the type \( A \rightarrow B \) and \( B \rightarrow C \), where \( A \) is candidate key, \( B \) is no candidate key and \( C \) contains at least one non-key attribute is forbidden.

The last condition is rather unintuitive but helps to ensure that every schema has a dependency-preserving decomposition into 3NF.

Example

- relation \( \text{lecture}(id, title, pers-id, room) \)
- Relation is not in 3NF because the FD \( \text{pers-id} \rightarrow \text{room} \) exists, and \( \text{pers-id} \) is not a key and \( \text{room} \) is not (part of) a candidate key.
Possible anomalies:
- Information about a professor and his/her room are not available without assignment of a lecture.
- Update anomaly: Change of the room number of a professor requires a change for each course with the same professor.
- Deletion anomaly: If a professor does not hold a class any more, all information about the professor and his/her room is removed from the database.

Solution: Splitting of the schema lecture into the two schemas lecture(id, title, pers-id) and Prof(pers-id, room).

Conclusion: The 3NF eliminates the dependencies from non-key attributes.

3NF synthesis algorithm

Goal: Decomposition of a relation schema \( R \) with the FDs \( F \) into relation schemas \( R_1, \ldots, R_n \) so that the following three criteria are fulfilled:
- \( R_1, \ldots, R_n \) is a lossless decomposition of \( R \).
- The decomposition preserves the FDs.
- The schemas \( R_1, \ldots, R_n \) each fulfil the 3NF.
synthesis algorithm for computing the decomposition on the basis of $F$:

- **step 1:** determine a canonical cover $F_c$ for $F$
  (i.e., left reduction of the FDs, right reduction of the remaining FDs, removal of FDs of the form $A \rightarrow \emptyset$, union rule for identical left sides)

- **step 2:** for each FD $A \rightarrow B \in F_c$:
  + create a relation schema $R_A := A \cup B$
  + assign the FDs $F_A = \{C \rightarrow D \in F_c \mid C \cup D \subseteq R_A\}$ to $R_A$

- **step 3:** If all schemas $R_A$ created in step 2 do not contain a candidate key of the original schema $R$, additionally create a relation with the schema $R_K = K$ and $F_K = \emptyset$ where $K$ is a candidate key of $R$.

- **step 4:** Eliminate schemas $R_A$ that are contained in another schema $R_{A'}$.

The result is not uniquely defined, since a set of FDs can have more than one canonical cover. In some cases the result of the algorithm depends on the order in which it considers the dependencies in $F_c$. 