The Chase Test for Lossless Join Decomposition (I)

- If $t \in \pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \ldots \bowtie \pi_{S_k}(R)$, then there must be tuples in $R$, say $t_1, t_2, \ldots, t_k$, such that $t$ is the join of the projections of each $t_i$ onto the sets of attributes $S_i$, for $i = 1, 2, 3, \ldots, k$.

- We draw a *tableau* and collect what we know.
  - Assuming $R$ has the attributes $A, B, \ldots$, we use $a, b, \ldots$ for the components of $t$.
  - For the $t_i$ we use the same letter as $t$ in the components that are in $S_i$.
  - We subscript the letter with $i$ if the component is not in $S_i$.

- Example 1: Assume a relation $R(A, B, C, D)$ is decomposed into the relations $S_1(A, D)$, $S_2(A, C)$, and $S_3(B, C, D)$. Then the tableau for this decomposition is as follows:

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d$</td>
</tr>
<tr>
<td>2</td>
<td>$a$</td>
<td>$b_2$</td>
<td>$c$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>$b$</td>
<td>$c$</td>
<td>$d$</td>
</tr>
</tbody>
</table>
The Chase Test for Lossless Join Decomposition (II)

- Example 1 (continued): Suppose the given FDs are $A \rightarrow B$, $B \rightarrow C$, and $CD \rightarrow A$.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>$b_2$</td>
<td>c</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

$A \rightarrow B$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>$b_1$</td>
<td>c</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

$B \rightarrow C$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>$b_1$</td>
<td>c</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

$CD \rightarrow A$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>$b_1$</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>$b_1$</td>
<td>c</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

The last row has become equal to $t$. We have shown that if $R$ satisfies the FDs $A \rightarrow B$, $B \rightarrow C$, and $CD \rightarrow A$, then whenever we project onto $\{A, D\}$, $\{A, C\}$, and $\{B, C, D\}$ and rejoin, what we get must have been in $R$. 
The Chase Test for Lossless Join Decomposition (III)

- Example 2: Consider the relation \( R(A, B, C, D) \) with the FD are \( B \rightarrow AD \) and the decomposition \( \{A, B\}, \{B, C\}, \) and \( \{C, D\} \). We obtain the tableaux

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b & c_1 & d_1 \\
a_2 & b & c & d_2 \\
a_3 & b_3 & c & d \\
\end{array}
\quad
\xrightarrow{B \rightarrow AD}
\quad
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b & c_1 & d_1 \\
a & b & c & d_1 \\
a_3 & b_3 & c & d \\
\end{array}
\]

There is no row that is fully unsubscripted. This decomposition does not have a lossless join.

- Another way to show this is: Treat above right table as a relation with three tuples. Then we obtain

\[
\begin{array}{cc}
A & B \\
\hline
a & b \\
a & b \\
a_3 & b_3 \\
\end{array}
\quad
\times
\quad
\begin{array}{cc}
B & C \\
\hline
b & c_1 \\
b & c \\
b_3 & c \\
\end{array}
\quad
\times
\quad
\begin{array}{cc}
C & D \\
\hline
c_1 & d_1 \\
c & d_1 \\
c & d \\
\end{array}
\]

\[
\nequal\neq
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a & b & c_1 & d_1 \\
a & b & c & d_1 \\
a_3 & b_3 & c & d_1 \\
a_3 & b_3 & c & d \\
\end{array}
\]

\neq R
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 affects 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:

\[
\begin{array}{|c|c|c|}
\hline
\text{parents} & \text{children} \\
\hline
\text{father} & \text{mother} & \{} \text{Liza, Lucia} \} \\
\hline
\text{Ben} & \text{Martha} & \{} \text{Liza, Lucia} \} \\
\hline
\text{Ben} & \text{Maria} & \{} \text{Theo, Josef} \} \\
\hline
\text{John} & \text{Martha} & \{} \text{Cleo} \} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{parents} & \text{children} & \text{cage} \\
\hline
\text{father} & \text{mother} & \text{name} \\
\hline
\text{Ben} & \text{Martha} & \text{Liza} & 5 \\
\hline
\text{Ben} & \text{Maria} & \text{Lucia} & 3 \\
\hline
\text{Ben} & \text{Maria} & \text{Theo} & 3 \\
\hline
\text{John} & \text{Martha} & \text{Josef} & 1 \\
\hline
\text{John} & \text{Martha} & \text{Cleo} & 9 \\
\hline
\end{array}
\]
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 $X \to \{A\}$ holds: If attribute $A$ is not part of a key and $X$ is a key, then there is no FD $Y \to \{A\}$ 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 $X \to \{A\}$ 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} $\to \{name\}$ and {reg-id, id} $\to \{sem\}$
  - additional FDs: {reg-id} $\to \{name\}$ and {reg-id} $\to \{sem\}$
  $\Rightarrow$ 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 $R_1, \ldots, R_n$ 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 \textit{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 \( lecture(id, title, pers-id, room) \)
- Relation is not in 3NF because the FD \( pers-id \rightarrow room \) exists, and \( pers-id \) is not a key and \( room \) 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 $R_1$, ..., $R_n$ so that the following three criteria are fulfilled:
- $R_1$, ..., $R_n$ is a lossless decomposition of $R$.
- The decomposition preserves the FDs.
- The schemas $R_1$, ..., $R_n$ 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 $F_c$. 