Example for decomposition algorithm

- relation schema $ProfAddr$ (pers-id, name, rank, room, city, street, zipcode, area-code, state, government)

- assumptions:
  - A city denotes the residence of a professor.
  - Government is the party of the president.
  - City names are unique within a state.
  - The zipcode does not change within a street.
  - Cities and streets lie completely in the single states.
  - A professor has exactly one office that he does not share.

- \{pers-id\} and \{room\} are candidate keys of the relation $ProfAddr$. The relation is not in 3NF since, e.g., the FD \{city, state\} $\rightarrow$ \{area-code\} violates the 3NF.
- step 1: computation of a canonical cover (precomputed)
  - FD 1: \{pers-id\} \rightarrow \{name, rank, room, city, street, state\}
  - FD 2: \{room\} \rightarrow \{pers-id\}
  - FD 3: \{city, street, state\} \rightarrow \{zipcode\}
  - FD 4: \{city, state\} \rightarrow \{area-code\}
  - FD 5: \{state\} \rightarrow \{government\}
  - FD 6: \{zipcode\} \rightarrow \{city, state\}

- step 2
  - from FD 1 we obtain:
    + \{pers-id, name, rank, room, city, street, state\}
    + FD 1 and FD 2 are assigned.
  - from FD 2 we obtain:
    + \{room, pers-id\}
    + FD 2 is assigned.
  - from FD 3 we obtain:
    + \{city, street, state, zipcode\}
    + FD 3 and FD 6 are assigned.
− from FD 4 we obtain:
  + {city, state, area-code}
  + FD 4 is assigned.
− from FD 5 we obtain:
  + {state, government}
  + FD 5 is assigned.
− from FD 6 we obtain:
  + {zipcode, city, state}
  + FD 6 is assigned.

 ques
− step 3
 Both room and pers-id are candidate keys of the original schema ProfAddr and are contained in a schema $R_A$.
− step 4
  − \{room, pers-id\} ⊆ \{pers-id, name, rank, room, city, street, state\}
  − \{zipcode, city, state\} ⊆ \{city, street, state, zipcode\}

 ques
 We obtain: \{(pers-id, name, rank, room, city, street, state), \{FD 1, FD 2\}\}, \{(city, street, state, zipcode), \{FD 3, FD 6\}\}, \{(city, state, area-code), \{FD 4\}\}, \{(state, government), \{FD 5\}\}
How to Determine All Candidate Keys for a Given Relation and a Set of FDs (I)

Example: Let $R(A, B, C, D, E, F)$ be a relation schema, and let $S = \{DF \rightarrow C, BC \rightarrow F, E \rightarrow A, ABC \rightarrow E\}$ be a set of FDs. Determine all candidate keys.

Do you have any idea what the candidate keys of $R$ are?

No? No idea?

Me neither.

Do you have any idea how many candidate keys $R$ has?

No? No idea?

Me neither.

Conclusion: Systematic consideration needed for this problem.
How to Determine All Candidate Keys for a Given Relation and a Set of FDs (II)

**Step 1:** *Determine the attributes that are neither on the left side nor on the right side of any FD.*

*Reason:* These attributes cannot be generated by any FD. Therefore, they must belong to any candidate key to generate themselves by reflexivity.

**Step 2:** *Determine the attributes that are only on the left side of any FD.*

*Reason:* These attributes can never be reached by any FD since there is no FD that has them on their right side. Therefore, they must also belong to any candidate key in order to generate themselves by reflexivity.

**Step 3:** *Determine the attributes that are only on the right side of any FD.*

*Reason:* These attributes can only be reached by other attributes or attribute combinations. That is, they cannot be part of any candidate key.

**Step 4:** *Combine the attributes from steps 1 and 2.*

*Reason:* Any candidate key must contain them.
How to Determine All Candidate Keys for a Given Relation and a Set of FDs (III)

**Step 5:** Compute the closure of the attributes from Step 4. If the attribute closure contains all attributes of $R$, then the attributes from Step 4 form the only candidate key, and the algorithm terminates.

**Reason:** No matter how many candidate keys there are, every one of them must contain these attributes, and they already reach all attributes in $R$. Hence they form the only key.

**Step 6:** Otherwise, determine the attributes that are included neither in Step 3 nor in Step 4.

**Reason:** The attributes from Step 3 cannot be part of any candidate key. The attributes from Step 4 have already been identified as parts of any candidate key. The remaining attributes are those that appear on both sides of an FD.

**Step 7:** Compute the closure of the attributes from Step 4 (if there are any) and every possible combination of attributes from Step 6, and determine those minimal attribute closures that are equal to $R$.

**Reason:** We have to find possible combinations of attributes or single missing attributes so that all attributes in $R$ can be reached from the resulting attribute sets.
How to Determine All Candidate Keys for a Given Relation and a Set of FDs (IV)

Back to the example: Let $R(A, B, C, D, E, F)$ be a relation schema, and let $S = \{DF \rightarrow C, BC \rightarrow F, E \rightarrow A, ABC \rightarrow E\}$ be a set of FDs. Determine all candidate keys.

Step 1: None.
Step 2: $BD$
Step 3: None.
Step 4: $BD$
Step 5: The closure of $BD$ is $BD$, that is, $BD^+ = BD$. This does not include all attributes from $R$, and therefore we have to continue with the next step.
Step 6: $ACEF$
Step 7: Consider the sets with three attributes: $BDA^+ = ABD, BDC^+ = BCDF, BDE^+ = ABDE, BDF^+ = BCDF$. No candidate keys found.

Consider the sets with four attributes: $BDAC^+ = ABCDEF, BDAE^+ = ABDE, BDAF^+ = ABCDEF, BDCE^+ = ABCDEF, BDCF^+ = BCDF, BDEF^+ = ABCDEF$. Candidate keys found: $ABCD, ABDF, BCDE, BDEF$

Consider the sets with five attributes for $BDAE$ and $BDCF$: $BDAEC^+ = BDAC^+, BDAEF^+ = BDEF^+, BDCF^+ = BDAF^+, BDCF^+ = BDCE^+$. No further candidate keys found. Done!
How to Determine All Candidate Keys for a Given Relation and a Set of FDs (V)

Same example but different presentation: Let $R(A, B, C, D, E, F)$ be a relation schema, and let $S = \{DF \rightarrow C, BC \rightarrow F, E \rightarrow A, ABC \rightarrow E\}$ be a set of FDs. Determine all candidate keys.

Draw table with three columns (this corresponds to steps 1, 2, 3, 4, 6): L = Attributes on left sides only, R = Attributes on right sides only, B = Attributes on both sides

<table>
<thead>
<tr>
<th>L</th>
<th>B</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B, D$</td>
<td>$A, C, E, F$</td>
<td>-</td>
</tr>
</tbody>
</table>

Perform steps 5 and 7.
7.6 Boyce-Codd Normal Form

- A relation schema $R$ with FDs $F$ is in **Boyce-Codd normal form (BCNF)**, if, and only if, it is in 3NF 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 $R$.

- conclusion: The BCNF eliminates dependencies among attributes that are part of a candidate key.

- example:
  - $CarIndex(manufacturer, manufacturer-id, model-id)$
  - consider FDs:
    + $FD1: \{model-id, manufacturer\} \rightarrow \{manufacturer-id\}$
    + $FD2: \{manufacturer-id\} \rightarrow \{manufacturer\}$

- example is in 3NF, but not in BCNF
The following anomalies can arise:
- Insertion of the same manufacturer with different manufacturer ids (and different model ids) is possible.
- 1:1-relationship between manufacturer and manufacturer-id is connected to model-id.

Properties of a schema in BCNF
- A relation schema $R$ with associated FDs $F$ can be decomposed into relation schemas $R_1, \ldots, R_n$ so that holds:
  + The decomposition is lossless.
  + The schemas $R_i (1 \leq i \leq n)$ are all in BCNF.
- But: We cannot always find a BCNF decomposition which is also dependency-preserving. This case is seldom in practice.

Procedure
- Decomposition of a schema from 3NF to BCNF
- Check if this decomposition is dependency-preserving. If this is the case, take this schema. Otherwise use the original schema in 3NF.
- Example: Producer(manufacturer-id, manufacturer), CarIndexNew(manufacturer-id, model-id), decomposition maintains FD2 but loses FD1 lossless join decomposition since CarIndex = Producer $\Join$ CarIndexNew
8. Application Programming

8.1 Introduction

Database and programming languages

- SQL is a powerful declarative query language. The formulation of queries in SQL is usually simpler than the coding of the same queries in an all-purpose programming language. There are at least two reasons for accessing a database with a programming language from a user perspective:
  - Not all queries can be expressed in SQL (little functionality for “everyday” programming) since SQL does not have the full expressive power of a programming language. In order to be able to express such queries, SQL can be embedded into a more powerful language. Applications are usually developed in imperative and object-oriented languages (C, Cobol, Fortran, Java, C++, ...).
  - Non-declarative actions like printing, interaction with the user, or transmission of query results to a graphical user interface are outside of SQL. Task sharing: query processing and updates with SQL, all other tasks with the aid of a programming language.
SQL queries can be automatically optimized and efficiently executed. The use of a programming language only makes an automatic optimization extraordinarily difficult.

Ad hoc queries are posed mostly by experts and more seldom. Frequently non-inter-active batch applications are needed. Often the possibilities of the DBMS for representing data are limited and unsuitable for user requirements.

Special integration problem (impedance mismatch):
- programming language supports the processing of single data records (tuple-oriented approach).
- SQL supports the processing of data records, i.e., of relations (set-oriented approach).

consequently the question: How can the programming of database tasks be combined with the “usual” tasks without abandoning the benefits of SQL?