Functional Dependencies
- Definition
- Functional Dependencies and Keys
- Inference Rules for Functional Dependencies
- Closure of Attribute Sets
- Canonical Cover

Normalization
- Goals
- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
RELATIONAL DATABASE DESIGN
AND FUNCTIONAL DEPENDENCIES

- Relational database design requires that we find a “good” collection of relation schemas.
  - A bad design may lead to several problems.

- Functional dependencies can be used to refine a relation schema reduced from an E-R schema by iteratively decomposing it (called normalization) to place it in a certain normal form.
  - The first four normal forms ⇒ use only functional dependencies.
  - Additional normal forms ⇒ use other types of dependencies

☞ Normal forms do not guarantee a good design!
Normalization decomposes unsatisfactory relation schemas into fragments (i.e., breaks them up into two or more relation schemas) that possess more desirable properties.

- Eliminates data redundancy and update anomalies, preserves dependencies and is lossless.

Normalization provides a series of tests for relation schemas.

- If a relation schema fails the test, then we need to decompose it.

Normalization is expressed in terms of normal forms:
- the first four normal forms \( \Rightarrow \) use only FDs
- additional normal forms \( \Rightarrow \) use other types of dependencies

Normal forms do not guarantee a good design!
NORMAlIZATION: GOALS

Design Guideline 1: Clear Semantics for Attributes
Design a relation schema so that it is easy to explain its meaning. Typically, this means that we should not combine attributes from multiple real-world entities in a single relation schema.

☞ Each relation schema should have a well-defined, unambiguous meaning.

Design Guideline 2: Minimize Use of Null Values
As far as possible, avoid placing attributes in relation schemas whose values may be null. If nulls are unavoidable, make sure that they apply in exceptional cases only.

☞ Null values lead to problems of understanding the meaning of attributes and specifying certain operations (e.g., aggregation operations).
**NORMAlIZATION: GOALS (cont'd)**

---

**Design Guideline 3: Minimize Redundancy**

Design relation schemas so that no insertion, deletion or update anomalies occur in the relation instances. If any update anomalies are present, note them clearly so that update programs will operate correctly.

---

☞ A relation schema has redundancy if there is an FD where the LHS is not a key.

☞ Redundant data in relations can cause operation anomalies.

---

**Operation Anomalies**

- **insertion** (e.g., insert license fee for engine size 5)
- **deletion** (e.g., delete instance “BMW, 7.35i, …”)
- **update** (e.g., update license fee for engine size 1)

---

<table>
<thead>
<tr>
<th>Car</th>
<th>make</th>
<th>model</th>
<th>engineSize</th>
<th>fee</th>
<th>origin</th>
<th>tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan Sunny</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fiat Mirafiori</td>
<td>1</td>
<td>4,000</td>
<td>Italy</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Honda Accord</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toyota Camry</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ford Mustang</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ford Mustang</td>
<td>2</td>
<td>5,000</td>
<td>U.S.A.</td>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMW 7.35i</td>
<td>3</td>
<td>6,000</td>
<td>Germany</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toyota Camry</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Design Guideline 4: **Lossless Decomposition**

The normalized relation schemas should contain the same information as the original schema; otherwise, decomposition results in information loss.

- A decomposition is lossless (aka lossless join) if the original relation instance can be recovered from the schema fragments.
  - Joining all the fragments results in exactly the original relation instance.

- In general, a decomposition of $R$ into $R_1$ and $R_2$ is lossless if and only if at least one of the following FDs is in $F^+$:

  $$R_1 \cap R_2 \rightarrow R_1 \quad \text{and} \quad R_1 \cap R_2 \rightarrow R_2$$

  ❨ The common attributes of $R_1$ and $R_2$ must be a superkey for $R_1$ or $R_2$. ❩
### LOSSY DECOMPOSITION EXAMPLE

Decompose $R(A, B, C)$ into $R_1(A, B)$ and $R_2(B, C)$.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>$m$</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>$n$</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>$p$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$m$</td>
</tr>
<tr>
<td>2</td>
<td>$n$</td>
</tr>
<tr>
<td>1</td>
<td>$p$</td>
</tr>
</tbody>
</table>

The decomposition is lossy since the join produces two extra tuples. Thus, the decomposition “loses” some information! Note that the common attribute $B$ is not a superkey of either $R_1$ or $R_2$. 

**The decomposition is lossy** since the join produces two extra tuples. Thus, the decomposition “loses” some information! Note that the common attribute $B$ is not a superkey of either $R_1$ or $R_2$. 

DSAA 5012
L10: RELATIONAL DATABASE DESIGN
**Design Guideline 5: Preserve Functional Dependencies**

As far as possible, functional dependencies should be preserved within each relation schema; otherwise, checking updates for violation of functional dependencies may require computing joins, which is expensive.

- Functional dependencies represent real-world constraints.

- If a functional dependency does not appear in any relation schema (i.e., it is “lost”), the constraint may be much more difficult to enforce.

- The decomposition of a relation schema $R$ with FDs $F$ is a set of schema fragments $R_i$ with FDs $F_i$.
  - $F_i$ is the subset of dependencies in $F^+$ (the closure of $F$) that involves only attributes in $R_i$.

- The decomposition is dependency preserving if and only if $(\bigcup F_i)^+ = F^+$.
  - Every FD in $F$ is present in some fragment $R_i$. 

© DSAA 5012
DECOMPOSITION EXAMPLE: NON-DEPENDENCY PRESERVING

R(A, B, C)  Key: A  \( F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\} \)  \( F = F^+ \)

For FD \( B \rightarrow C \), LHS is not a key \( \implies \) \( R \) can have considerable redundancy.

Solution: Break \( R \) into relations \( R_1(A, B) \), \( R_2(A, C) \) (normalization).

\[
\begin{array}{c|c|c}
A & B & C \\
\hline
1 & 2 & 3 \\
2 & 2 & 3 \\
3 & 2 & 3 \\
4 & 2 & 4 \\
\end{array}
\]

\[
\begin{array}{c|c}
A & B \\
\hline
1 & 2 \\
2 & 2 \\
3 & 2 \\
4 & 2 \\
\end{array}
\]

\[
\begin{array}{c|c}
A & C \\
\hline
1 & 3 \\
2 & 3 \\
3 & 3 \\
4 & 4 \\
\end{array}
\]

The decomposition is lossless since the common attribute \( A \) is a key for \( R_1 \) (and \( R_2 \)).

The decomposition is not dependency preserving because \( F_1 = \{A \rightarrow B\}, F_2 = \{A \rightarrow C\} \) and \( (F_1 \cup F_2)^+ \neq F^+ \). The FD \( B \rightarrow C \) is lost.

In practice, each “lost” FD is implemented as an assertion (a type of constraint), which is checked when there are updates. Thus, to find violations on \( B \rightarrow C \), \( R_1 \) and \( R_2 \) must be joined, which can be very expensive.
DECOMPOSITION EXAMPLE: DEPENDENCY PRESERVING

R(A, B, C)  Key: A  \( F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\} \)  \( F = F^+ \)

Break R into relations \( R_1(A, B) \), \( R_2(B, C) \).

The decomposition is **lossless** since the common attribute B is a key for \( R_2 \).

The decomposition is **dependency preserving** because \( F_1 = \{A \rightarrow B\} \), \( F_2 = \{B \rightarrow C\} \) and \((F_1 \cup F_2)^+ = F^+ \) (since \( A \rightarrow B, B \rightarrow C \models A \rightarrow C \)).

Violations of the FDs can be found by inspecting the individual tables, **without** performing a join.

☞ How a relation is decomposed, may determine whether functional dependencies are preserved.
RELATIONAL DATABASE DESIGN: NORMALIZATION

EXERCISES 1, 2
EXERCISE 1

Given: \( R(A, B, C, D, E) \) \hspace{1cm} F = \{A \rightarrow BC\}

Decomposition: \( R_1(A, B, C) \) and \( R_2(A, D, E) \)

a) Is the decomposition lossless? Why? \( (\text{iff } R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2) \)

Yes The common attribute \( A \) is a key for \( R_1 \).

b) Is the decomposition dependency preserving? Why? \( (\text{iff } (\cup F_i)^+ = F^+) \)

Yes \( A \rightarrow BC \) is preserved in \( R_1 \).

c) Is the decomposition \( R_1(A, B, C) \) and \( R_2(C, D, E) \) lossless? Why?

No \( C \) is not a key for any table.
EXERCISE 2

Given:  \( R(A, B, C, D, E) \) \[ F = \{A \rightarrow BC, \ CD \rightarrow E, \ B \rightarrow D, \ E \rightarrow A\} \]

Decomposition: \( R_1(A, B, C) \) and \( R_2(A, D, E) \)

a) Is the decomposition lossless? \( (\text{iff } R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2) \)

   Yes  The common attribute A is a key for \( R_1 \).

b) Is the decomposition dependency preserving? \( (\text{iff } (\cup F_i)^+ = F^+) \)

   No    We lose \( CD \rightarrow E \) and \( B \rightarrow D \).
Prime and non-prime attributes

An attribute is a prime attribute if it is part of any candidate key. Otherwise, it is a non-prime attribute.

Partial dependency visualization

Transitive dependency visualization
# NORMALIZATION:

## EXAMPLE RELATION SCHEMA & DATABASE

**Car**

<table>
<thead>
<tr>
<th>make</th>
<th>model</th>
<th>engineSize</th>
<th>fee</th>
<th>origin</th>
<th>tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan</td>
<td>Sunny</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Fiat</td>
<td>Mirafiori</td>
<td>1</td>
<td>4,000</td>
<td>Italy</td>
<td>85</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>2</td>
<td>5,000</td>
<td>U.S.A.</td>
<td>75</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
<td>3</td>
<td>6,000</td>
<td>Germany</td>
<td>95</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
</tbody>
</table>

**Functional Dependencies**

- make, model, engineSize → origin
- make, model, engineSize → tax
- make, model, engineSize → fee
- origin → tax
- engineSize → fee
- origin → engineSize

**FD visualization**

- due to the primary key from real-world knowledge
FIRST NORMAL FORM (1NF)

A relation schema is in *First Normal Form (1NF)* if all attributes are atomic (single-valued).

☞ There are no multi-valued or composite attributes.

- Relation schemas are always in 1NF according to the definition of the relational model and according to our strategy for reducing an E-R schema to relation schemas.
SECOND NORMAL FORM (2NF)

A relation schema is in Second Normal Form (2NF) if all non-prime attributes are fully functionally dependent on every candidate key.

- $R$ is a relation schema, with the set $F$ of FDs.

- $R$ is in 2NF if and only if

  For each FD: $X \rightarrow A$ in $F^+$:
  
  - $A \in X$ (the FD is trivial) or
  - $X$ is not a proper subset of a candidate key for $R$ or
  - $A$ is a prime attribute for $R$.

☞ A subset of a candidate key cannot determine a non-prime attribute.
SECOND NORMAL FORM (2NF) EXAMPLE

- *make, model, engineSize* is a candidate key (it is not a proper subset).
- *engineSize* is a proper subset of a candidate key.
- *fee* is a non-prime attribute.
- Hence, the relation schema is **not in 2NF** due to the FD \( \text{engineSize} \rightarrow \text{fee} \).

Note redundancy.

<table>
<thead>
<tr>
<th>make</th>
<th>model</th>
<th>engineSize</th>
<th>fee</th>
<th>origin</th>
<th>tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan</td>
<td>Sunny</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Fiat</td>
<td>Mirafiori</td>
<td>1</td>
<td>4,000</td>
<td>Italy</td>
<td>85</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>4</td>
<td>7,000</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>2</td>
<td>5,000</td>
<td>U.S.A.</td>
<td>75</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
<td>3</td>
<td>6,000</td>
<td>Germany</td>
<td>95</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>1</td>
<td>4,000</td>
<td>Japan</td>
<td>90</td>
</tr>
</tbody>
</table>

Recall Operation Anomalies

- **insertion** (e.g., insert license fee for engine size 5)
- **deletion** (e.g., delete instance “BMW, 7.35i, …”)
- **update** (e.g., update license fee for engine size 1)
SECOND NORMAL FORM (2NF) EXAMPLE (cont’d)

FDs in original schema

FDs in 2NF schemas

partial dependency; fee is non-prime

decompose the schema

decompose the tables

The decomposition resolves the previous anomalies.

Car

<table>
<thead>
<tr>
<th>make</th>
<th>model</th>
<th>engineSize</th>
<th>origin</th>
<th>tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan</td>
<td>Sunny</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Fiat</td>
<td>Mirafiori</td>
<td>1</td>
<td>Italy</td>
<td>85</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>4</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>4</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>2</td>
<td>U.S.A.</td>
<td>75</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
<td>3</td>
<td>Germany</td>
<td>95</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
</tbody>
</table>

Licensing

<table>
<thead>
<tr>
<th>engineSize</th>
<th>fee</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4000</td>
</tr>
<tr>
<td>2</td>
<td>5000</td>
</tr>
<tr>
<td>3</td>
<td>6000</td>
</tr>
<tr>
<td>4</td>
<td>7000</td>
</tr>
</tbody>
</table>
EXERCISE 3

a) Given: \( R(A, B, C, D) \) \( F = \{AB \rightarrow CD, B \rightarrow C\} \)

Is \( R \) in 2NF? Why?

Key: \( AB \) \( AB^+ = \{A, B, C, D\} \) \( B^+ = \{B, C\} \)

No  For \( B \rightarrow C \), \( B \) is a proper subset of the key \( AB \) and \( C \) is non-prime.
So, \( R \) is not in 2NF.

b) Given: \( R(A, B, C, D) \) \( F = \{AB \rightarrow CD, C \rightarrow D\} \)

Is \( R \) in 2NF? Why?

Key: \( AB \) \( AB^+ = \{A, B, C, D\} \) \( C^+ = \{C, D\} \)

Yes  For \( C \rightarrow D \), \( C \) is not a proper subset of the key, so \( R \) is in 2NF.

2NF
\( R \) is in 2NF if and only if
For each FD: \( X \rightarrow A \) in \( F^+ \):
\( A \in X \) (trivial FD) or
\( X \) is not a proper subset of a candidate key for \( R \) or
\( A \) is a prime attribute for \( R \).
THIRD NORMAL FORM (3NF)

A relation schema is in Third Normal Form (3NF) if it is in 2NF and every non-prime attribute is non-transitively dependent on every candidate key.

- \( R \) is a relation schema, with set \( F \) of FDs.
- \( R \) is in 3NF if and only if
  - For each FD: \( X \rightarrow A \) in \( F^+ \):
    - \( A \in X \) (trivial FD) or
    - \( X \) is a superkey for \( R \) or
    - \( A \) is a prime attribute for \( R \).
  - For every FD that does not contain extraneous attributes either:
    - the LHS is a candidate key, or
    - the RHS is a prime attribute (i.e., it is part of some candidate key).
THIRD NORMAL FORM (3NF) EXAMPLE

– For the FD \( \text{origin} \rightarrow \text{tax} \), \( \text{origin} \) is not a superkey.
– \( \text{tax} \) is not a prime attribute.
– Hence, the relation schema is not in 3NF due to the FD \( \text{origin} \rightarrow \text{tax} \).

<table>
<thead>
<tr>
<th>make</th>
<th>model</th>
<th>engineSize</th>
<th>origin</th>
<th>tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan</td>
<td>Sunny</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Fiat</td>
<td>Mirafiori</td>
<td>1</td>
<td>Italy</td>
<td>85</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>4</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>4</td>
<td>Canada</td>
<td>50</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>2</td>
<td>U.S.A.</td>
<td>75</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
<td>3</td>
<td>Germany</td>
<td>95</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>1</td>
<td>Japan</td>
<td>90</td>
</tr>
</tbody>
</table>

Note redundancy.

Operation Anomalies

- **insertion** (e.g., insert tax rate 40 for Australia)
- **deletion** (e.g., delete instance “BMW, 7.35i, …”)
- **update** (e.g., update tax rate for origin Japan)
THIRD NORMAL FORM (3NF) EXAMPLE (cont’d)

FDs in 2NF schemas

- make 
motor 
engineSize

tax 
norigin 
fee

transitive dependency; tax is non-prime

decompose the schema

FDs in 3NF schemas

- make 
motor 
engineSize

origin 
fee

decompose the schema

The decomposition resolves the previous anomalies.

If none of the decomposed relations contains a candidate key of the original relation, then add a relation containing one of the candidate keys.
3NF DECOMPOSITION ALGORITHM

Let \( R \) be the initial relation schema with FDs \( F \).
Compute a canonical cover \( F_c \) of \( F \).
\[
S = \emptyset \quad \text{(S is a set of relation schemas)}
\]
For each FD \( X \rightarrow Y \) in the canonical cover \( F_c \),
\[
S = S \cup (X, Y) \quad \text{(for each FD create a relation schema; add it to S)}
\]
If no schema contains a candidate key for \( R \),
Choose any candidate key \( K \)
\[
S = S \cup K \quad \text{(add any candidate key as a relation schema)}
\]

☞ The algorithm always creates a lossless-join, dependency preserving, 3NF decomposition.

(Also called the 3NF Synthesis Algorithm.)
RELATIONAL DATABASE DESIGN: NORMALIZATION

EXERCISE 4
EXERCISE 4

Identify the candidate key(s) and the current highest normal form for each of the following relation schemas given their corresponding FDs.

a) $R(A, B, C, D, E)$  
   \[ F = \{A \rightarrow B, C \rightarrow D\} = F^+ \]
   \[ A^+ = \{A, B\} \quad C^+ = \{C, D\} \]

   Candidate keys: ACE

   \[ \Rightarrow \text{ For } A \rightarrow B \text{ and } C \rightarrow D \]
   i. A and C are proper subsets of the candidate key ACE (both FDs fail 1st 2NF test).
   ii. both B and D are not prime attributes of R (both FDs fail 2nd 2NF test).

   \[ \text{☞ Both FDs violate 2NF.} \]

   Normal form: 1NF

---

**2NF**

R is in 2NF if and only if
For each FD: $X \rightarrow A$ in $F^+$:
- $A \in X$ (trivial FD) or
- $X$ is not a proper subset of a candidate key for $R$ or
- $A$ is a prime attribute for $R$.

**3NF**

R is in 3NF if and only if
For each FD: $X \rightarrow A$ in $F^+$:
- $A \in X$ (trivial FD) or
- $X$ is a superkey for $R$ or
- $A$ is a prime attribute for $R$. 

EXERCISE 4 (cont’d)

Identify the candidate key(s) and the **current highest normal form** for each of the following relation schemas given their corresponding FDs.

b) $R(A, B, C)$

$F = \{ AB \rightarrow C, \ C \rightarrow B \} = F^+$

$AB^+ = \{ A, B, C \}$

$C^+ = \{ C, B \}$

**Candidate keys:** $AB, AC$

⇒ For $AB \rightarrow C$, 
  $C$ *is* a prime attribute of $R$ 
  (FD passes $2^{nd}$ 2NF and 3NF tests).

⇒ For $C \rightarrow B$, 
  $B$ *is* a prime attribute of $R$ 
  (FD passes $2^{nd}$ 2NF and 3NF tests).

☞ Both FDs satisfy **3NF**.

Normal form: **3NF**
EXERCISE 4 (cont’d)

Identify the candidate key(s) and the current highest normal form for each of the following relation schemas given their corresponding FDs.

c) R(A, B, C, F)

\[ F = \{AB \rightarrow C, C \rightarrow F\} = F^+ \]

\[ AB^+ = \{A, B, C, F\} \quad C^+ = \{C, F\} \]

Candidate keys: AB

\[ \Rightarrow \text{For } AB \rightarrow C \]

i. AB is not a proper subset of a candidate key (FD passes 1st 2NF test);
ii. AB is a superkey for R (FD passes 1st 3NF test).

\[ \Rightarrow \text{For } C \rightarrow F \]

i. C is not a proper subset of a candidate key (FD passes 1st 2NF test);
ii. C is not a superkey of R (FD fails 1st 3NF test);
iii. F is not a prime attribute (FD fails 2nd 3NF test).

☞ Both FDs satisfy 2NF.

Normal form: 2NF
BOYCE-CODD NORMAL FORM (BCNF)

A relation schema is in *Boyce-Codd Normal Form (BCNF)* if every determinant (left hand side) of its FDs is a superkey.

- R is a relation schema, with the set \( F \) of FDs.
- R is in BCNF *if and only if*
  
  For each FD: \( X \rightarrow A \) in \( F^+ \):
  
  \( A \in X \) *(trivial FD)* or
  
  X is a superkey for R.

☞ For every FD that does not contain extraneous attributes, the LHS is a candidate key.

➢ BCNF tables have no redundancy (that can be removed by FDs).

➢ If a table is in BCNF it is also in 3NF (and 2NF and 1NF).
BOYCE-CODD NORMAL FORM (BCNF): EXAMPLE

- For the FD \texttt{origin} \rightarrow \texttt{engineSize}, \texttt{origin} is not a superkey.
- Hence, this relation schema is not in BCNF due to the FD \texttt{origin} \rightarrow \texttt{engineSize}.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{make} & \textbf{model} & \textbf{engineSize} & \textbf{origin} \\
\hline
Nissan & Sunny & 1 & Japan \\
Fiat & Mirafiori & 1 & Italy \\
Honda & Accord & 1 & Japan \\
Toyota & Camry & 4 & Canada \\
Ford & Mustang & 4 & Canada \\
Ford & Mustang & 2 & U.S.A. \\
BMW & 7.35i & 3 & Germany \\
Toyota & Camry & 1 & Japan \\
\hline
\end{tabular}
\end{table}

\textbf{Note:} Need to use \texttt{null} values if we want to represent an engine size and origin, but do not know the make and model.

\textbf{Operation Anomalies}

- \textbf{insertion} (e.g., insert engine size 5 from Korea)
- \textbf{deletion} (e.g., delete all instances “Ford, Mustang, ...”)
- \textbf{update} (e.g., update engine size for Japan)
BOYCE-CODD NORMAL FORM (BCNF): **EXAMPLE** (cont’d)

FDs in 3NF schema

<table>
<thead>
<tr>
<th>make</th>
<th>model</th>
<th>origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nissan</td>
<td>Sunny</td>
<td>Japan</td>
</tr>
<tr>
<td>Fiat</td>
<td>Miraflori</td>
<td>Italy</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Japan</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>Canada</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>Canada</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
<td>U.S.A.</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
<td>Germany</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
<td>Japan</td>
</tr>
</tbody>
</table>

Engine sizes:

<table>
<thead>
<tr>
<th>origin</th>
<th>engineSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>1</td>
</tr>
<tr>
<td>Canada</td>
<td>4</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>2</td>
</tr>
<tr>
<td>Germany</td>
<td>3</td>
</tr>
<tr>
<td>Japan</td>
<td>1</td>
</tr>
</tbody>
</table>

This decomposition avoids the 3NF problems of redundancy and null values.
• We can generate the original relation instance by joining the two fragments, using a full outer join.

<table>
<thead>
<tr>
<th>Car</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>make</td>
<td>model</td>
</tr>
<tr>
<td>Nissan</td>
<td>Sunny</td>
</tr>
<tr>
<td>Fiat</td>
<td>Mirafiori</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
</tr>
<tr>
<td>Ford</td>
<td>Mustang</td>
</tr>
<tr>
<td>BMW</td>
<td>7.35i</td>
</tr>
<tr>
<td>Toyota</td>
<td>Camry</td>
</tr>
</tbody>
</table>

Is the decomposition dependency preserving?

No. We lose the FD \(\text{make}, \text{model}, \text{engineSize} \rightarrow \text{origin}\).

Can we have a dependency preserving decomposition?

No. However, the relation schema is decomposed, \(\text{make}, \text{model}, \text{engineSize} \rightarrow \text{origin}\) is lost since it involves all the attributes of the original 3NF Car relation.

A relation may not have a dependency preserving BCNF decomposition!
Let $R$ be the initial relation schema with set of FDs $F$.
Compute $F^+$
$S = R$
Until all relation schemas in $S$ are in BCNF
  For each $R$ in $S$
    For each FD $X \rightarrow Y$ that violates BCNF for $R$
      $S = (S - R) \cup (R - Y) \cup (X, Y)$
End until

- When a relation schema $R$ with BCNF violation $X \rightarrow Y$ is found:
  1. Remove $R$ from $S$, the set of relation schemas.
  2. Add a schema that has the same attributes as $R$ except for $Y$ (i.e., remove the RHS of the FD from $R$).
  3. Add a second schema that contains the attributes in $X$ and $Y$ (i.e., create a relation schema for the FD $X \rightarrow Y$).
BCNF DECOMPOSITION EXAMPLE

Given: \( R(A, B, C, D, E) \) \( F = \{A \rightarrow BE, C \rightarrow D\} \) (Note: \( F = F^+ \))
Candidate key: AC

- Both functional dependencies violate BCNF because the LHS is not a candidate key.

- Pick \( A \rightarrow BE \)
  - We can also choose \( C \rightarrow D \) \( \implies \) different choices may lead to different decompositions.

- \( R(A, B, C, D, E) \) generates: \( R_1(A, C, D) \) (remove the RHS of \( A \rightarrow BE \) from \( R \))
  \( R_2(A, B, E) \) (based on the FD \( A \rightarrow BE \))

Do we need to decompose further? Yes Why?
BCNF DECOMPOSITION EXAMPLE (cont’d)

We have: $R_1(\text{A, C, D})$ and $R_2(\text{A, B, E})$

$F = \{ \text{A} \rightarrow \text{BE}, \text{C} \rightarrow \text{D} \}$  (Note: $F = F^+$)

Candidate key: AC

• We need to decompose $R_1(\text{A, C, D})$ because of the FD $\text{C} \rightarrow \text{D}$.

• Thus $R_1(\text{A, C, D})$ is replaced with $R_3(\text{A, C})$ and $R_4(\text{C, D})$.

• Final decomposition: $R_2(\text{A, B, E})$, $R_3(\text{A, C})$, $R_4(\text{C, D})$. 
Final decomposition: \[ R_2(A, B, E) \quad R_3(A, C) \quad R_4(C, D) \]

FDs:
\[ F_2 = \{A \rightarrow BE\} \quad F_3 = \emptyset \quad F_4 = \{C \rightarrow D\} \]

Is the decomposition lossless?

Yes the algorithm always creates lossless decompositions.

In step \( S = (S - R) \cup (R - Y) \cup (X, Y) \) we replace \( R \) with relations \( (R - Y) \) and \( (X, Y) \) that have \( X \) as the common attribute and \( X \rightarrow Y \) (i.e., \( X \) is the key of \( (X, Y) \)).

Is the decomposition dependency preserving?

Yes because \( (F_2 \cup F_3 \cup F_4)^+ = F^+ \)

☞ But remember: sometimes dependencies cannot be preserved.
Functional Dependencies

- FDs are constraints derived from the application domain.
- FDs can be used to refine a relation schema reduced from an E-R schema.

Normalization

- When an E-R schema is not well designed, the relation schemas generated from it may have undesirable properties (update anomalies).
- Using functional dependencies, normalization remove these update anomalies by decomposing a relation schema into normal forms.
- While BCNF is the "best" normal form, it may not be dependency preserving.
- There is always a dependency preserving 3NF decomposition and, in practice, 3NF is often "good enough" for most applications.
COMP 3311: SYLLABUS

✓ Introduction
✓ Entity-Relationship (E-R) Model and Database Design
✓ Relational Algebra
✓ Structured Query Language (SQL)
✓ Relational Database Design

Storage and File Structure

- Indexing
- Query Processing
- Query Optimization
- Transactions
- Concurrency Control
- Recovery System
- NoSQL Databases
Upload your completed exercise worksheet to Canvas by 11 p.m. on March 5th.