Relational Normalization Theory

Chapter 6

Topics

- Limitations of ER modeling
  - Functional Dependencies
  - Normal Forms
  - Decompositions
  - Algorithms for BCNF and 3NF
  - Normalization & Performance

Limitations of E-R Designs

- Provides a set of guidelines, does not result in a unique database schema
- Does not provide a way of evaluating alternative schemas
- Normalization theory provides a mechanism for analyzing and refining the schema produced by an E-R design
  - Concepts and algorithms

Redundancy and Other Problems

- Set valued attributes in the E-R diagram result in multiple rows in corresponding table
- Example: Person (SSN, Name, Address, Hobbies)
  - A person entity with multiple hobbies yields multiple rows in table Person
    - Hence, the association between Name and Address for the same person is stored redundantly
  - SSN is key of entity set, but (SSN, Hobby) is key of corresponding relation
  - The relation Person can’t describe people without hobbies
Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>biking, hiking</td>
</tr>
<tr>
<td>2222</td>
<td>Kyle</td>
<td>4 Pine</td>
<td>{stamps, biking}</td>
</tr>
</tbody>
</table>

Anomalies

- Redundancy leads to anomalies:
  - Update anomaly: A change in Address must be made in several places
  - Deletion anomaly: Suppose a person gives up all hobbies. Do we:
    - Set Hobby attribute to null? No, since Hobby is part of key
    - Delete the entire row? No, since we lose other information in the row
  - Insertion anomaly: Hobby value must be supplied for any inserted row since Hobby is part of key
- Sometimes the term Update anomaly is used for as a generic term for all 3 types of anomaly

Redundancy without Set Valued Attributes

- Dependencies between attributes cause redundancy
  - Ex. All addresses in the same town have the same zip code (for a small enough town)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Town</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Joe</td>
<td>Stony Brok</td>
<td>11790</td>
</tr>
<tr>
<td>4321</td>
<td>Mary</td>
<td>Stony Brok</td>
<td>11790</td>
</tr>
<tr>
<td>5454</td>
<td>Tom</td>
<td>Stony Brok</td>
<td>11790</td>
</tr>
</tbody>
</table>

Removing redundancy: Decomposition

- Solution: use two relations to store Person information
  - Person1 (SSN, Name, Address)
  - Hobbies (SSN, Hobby)
- Decomposition removes redundancy
  - Yes, SSN repeated in Hobbies (more on next slide)
- No update anomalies:
  - Name and address stored once
  - A hobby can be separately supplied or deleted
  - People without hobbies can now be described
Problem with Repeated SSN?

- But SSN is repeated in Hobbies. If SSN changes, we will have to update multiple rows in Hobbies
  - How big a problem is it?

1. SSN is needed in Hobbies
   - Not all decompositions are equal
   - No redundancy if we decompose into 4 relations with 1 attribute each: SSN, Name, Address, Hobby. But the decomposition is meaningless

2. If SSN (or whatever attribute is the identifier) is static, then it would never be updated

Normalization Theory

- Result of E-R analysis need further refinement
- Appropriate decomposition can solve problems
  - But all decompositions are not created equal
- The underlying theory for decomposing relations is referred to as normalization theory
  - Decide whether a particular relation is in a “good” form
  - If not, decompose it into a set of relations that are in “good” form
- Normalization is based on
  - functional dependencies which are a generalization of the concept of key constraints
  - and other kinds of dependencies, e.g., multivalued dependencies

Topics

- Limitations of ER modeling
- Functional Dependencies
  - Definition
  - Properties of FDs
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
- Normalization & Performance

Functional Dependencies

- **Definition:** A functional dependency (FD) on a relation schema \( R \) is a constraint \( X \rightarrow Y \), where \( X \) and \( Y \) are subsets of attributes of \( R \).
- **Definition:** An FD \( X \rightarrow Y \) is satisfied in an instance \( r \) of \( R \) if for every pair of tuples, \( t \) and \( s \): if \( t \) and \( s \) agree on all attributes in \( X \) then they must agree on all attributes in \( Y \)
FDs: Examples

- In the relation schema:
  - Person: SSN, Name, Address, Hobby
- The following FDs are satisfied:
  - SSN → Name
  - SSN → Address
- But the following FDs are not satisfied:
  - SSN → Hobby
  - Hobby → Name

FDs – Example 2

- Consider a brokerage firm that:
  - allows multiple clients to share an account, but each account is managed from a single office and
  - a client can have no more than one account in an office
- HasAccount (AcctNum, ClientId, OfficeId)
  - keys are (ClientId, OfficeId), (AcctNum, ClientId)
  - ClientId, OfficeId → AcctNum
  - AcctNum → OfficeId
  - Thus, attribute values need not depend only on key values

FDs: Comments

- FDs are identified from the enterprise being modeled
- Author, Title → PublDate
  - Shakespeare’s Hamlet published in 1600
  - If multiple editions of a book (same Author and Title) are given different PublDate, then the above FD will not be hold in the corresponding schema
- It may happen that an instance of a relation satisfies some FDs accidentally
  - E.g., in our example, Name → Address
- But such satisfaction is merely accidental if such an FD is not part of the relation schema
  - It would be legal to add tuple where Name is the same but Address is different

FDs: Comments (cont’d)

- Key constraint is a special kind of functional dependency:
  - all attributes of relation occur on the right-hand side of the FD:
- E.g., Relation
  - Person: SSN, Address, Name, Hobby
  - is decomposed into
  - Person1: SSN, Address, Name and
  - Hobbies: SSN, Hobby
- Then Person1 satisfies the FD
  - SSN → SSN, Name, Address
Topics

- Limitations of ER modeling
- Functional Dependencies
  - Definition
  - Properties of FDs
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
- Normalization & Performance

Entailment, Closure, Equivalence

- Definition: If $F$ is a set of FDs on schema $R$ and $f$ is another FD on $R$, then $F$ entails $f$ if every instance $r$ of $R$ that satisfies every FD in $F$ also satisfies $f$
  - Ex: $F = \{A \rightarrow B, B \rightarrow C\}$ and $f$ is $A \rightarrow C$
    - If $Town \rightarrow Zip$ and $Zip \rightarrow AreaCode$ then $Town \rightarrow AreaCode$
  - Note: The term covers is also used as a synonym to entails
- Definition: The closure of $F$, denoted $F^+$, is the set of all FDs entailed by $F$
- Definition: $F$ and $G$ are equivalent if $F$ entails $G$ and $G$ entails $F$

Entailment (cont’d)

- Satisfaction, entailment, and equivalence are semantic concepts – defined in terms of the actual relations in the “real world.”
  - They define what these notions are, not how to compute them
- How to check if $F$ entails $f$ or if $F$ and $G$ are equivalent?
  - Develop an algorithmic, syntactic ways to compute these notions

Armstrong’s Axioms for FDs

- This is the syntactic way of computing/testing the various properties of FDs
  - Will use these as inference rules to test entailment
- Reflexivity: If $Y \subseteq X$ then $X \rightarrow Y$ (trivial FD)
  - Name, Address $\rightarrow$ Name
- Augmentation: If $X \rightarrow Y$ then $XZ \rightarrow YZ$
  - If $Town \rightarrow Zip$ then $Town, Name \rightarrow Zip, Name$
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
Proof of Reflexivity

- Let \( Y \) and \( X \) be two subsets of the attributes in a relation schema \( R \) such that \( Y \subseteq X \).
- Consider two tuples \( t_1 \) and \( t_2 \) in some relation instance \( r \) of \( R \) such that
  - \( t_1 [X] = t_2 [X] \)
- Then it must be the case that
  - \( t_1 [Y] = t_2 [Y] \) since \( Y \subseteq X \)
- Hence if \( Y \subseteq X \) then \( X \rightarrow Y \)

Proof of Augmentation

- We will prove this inference rule by contradiction
- Assume \( X \rightarrow Y \) holds in a relation instance \( r \) of \( R \) but \( XZ \rightarrow YZ \) does not hold
- For this to be true, there must be two tuples \( t_1 \) and \( t_2 \) in \( r \) such that:
  - i) \( t_1 [X] = t_2 [X] \)
  - ii) \( t_1 [Y] = t_2 [Y] \) (by \( X \rightarrow Y \))
  - iii) \( t_1 [XZ] = t_2 [XZ] \) (LHS of \( XZ \rightarrow YZ \))
  - iv) \( t_1 [YZ] \neq t_2 [YZ] \) (RHS of \( XZ \rightarrow YZ \))
- From i) and iii) we deduce that \( t_1 [Z] = t_2 [Z] \)
- and from ii) and iv) we deduce that \( t_1 [YZ] \neq t_2 [YZ] \)
  - which is a contradiction
- Hence it must be that if \( X \rightarrow Y \) holds \( XZ \rightarrow YZ \) also hold

Proof of Transitivity

- Assume the following FDs hold in a relation schema \( R \)
  - i) \( X \rightarrow Y \) and
  - ii) \( Y \rightarrow Z \)
- Consider two tuples \( t_1 \) and \( t_2 \) in some relation instance \( r \) of \( R \) such that
  - iii) \( t_1 [X] = t_2 [X] \)
- Then from i) and iii) we deduce
  - iv) \( t_1 [Y] = t_2 [Y] \)
- Furthermore from ii) and iv) we deduce
  - v) \( t_1 [Z] = t_2 [Z] \)
- Hence from iii) and v) we deduce that \( X \rightarrow Z \)

More Derivation Rules: 1

- If \( X \rightarrow Y \) and \( X \rightarrow Z \) are satisfied by a relation then \( X \rightarrow YZ \) is satisfied by that relation
- Proof:
  - Given \( X \rightarrow Y \) and \( X \rightarrow Z \)
    \( X \rightarrow XY \) Augmentation by \( X \)
    \( YX \rightarrow YZ \) Augmentation by \( Y \)
    \( X \rightarrow YZ \) Transitivity
- We have derived the Union Rule for FDs:
  - An FD can be formed with the union of the RHSs of FDs that have the same LHS
More Derivation Rules: 2

• If \( X \rightarrow YZ \) is satisfied by a relation then that relation also satisfies \( X \rightarrow Y \) and \( X \rightarrow Z \)

• Proof:
  – Given \( X \rightarrow YZ \)
    \[ YZ \rightarrow Y \] Reflexivity
    \[ X \rightarrow Y \] Transitivity
  \( X \rightarrow Z \) is derived a similar manner

• We now also have derived the Decomposition Rule for FDs:
  – Given an FD with multiple attributes on the RHS, we can derive an FD with LHS of the original FD and any subset of attributes on the RHS

Soundness & Completeness

• Axioms are sound:
  – If an FD \( f: X \rightarrow Y \) can be derived from a set of FDs \( F \) using the axioms, then \( f \) holds in every relation that satisfies every FD in \( F \).

• Axioms are complete:
  – If \( F \) entails \( f \), then \( f \) can be derived from \( F \) using the axioms

• A consequence of completeness is the following (naive) algorithm to determining if \( F \) entails \( f \):
  – Use the axioms in all possible ways to generate \( F^+ \) (the set of possible FD’s is finite so this can be done) and see if \( f \) is in \( F^+ \)

Attribute Closure

• Calculating attribute closure leads to a more efficient way of checking entailment

• The attribute closure of a set of attributes, \( X \), with respect to a set of functional dependencies, \( F \), (denoted \( X^+_F \)) is the set of all attributes, \( A \), such that \( X \rightarrow A \)
  – \( X^+_F \) is not necessarily the same as \( X^+_{\overline{F}} \) if \( F1 \neq F2 \)

• Attribute closure and entailment:
  – Algorithm: Given a set of FDs, \( F \), then \( X \rightarrow Y \) if and only if \( X^+_F \supseteq Y \)

Computation of Attribute Closure \( X^+_F \)

\[
\begin{align*}
closure & := X; \quad \text{// since } X \subseteq X^+_F \\
\text{repeat} \\
old & := closure; \\
\text{if} \text{ there is an FD } Z \rightarrow V \text{ in } F \text{ such that } \\
\text{Z} & \subseteq \text{closure and } V \notin \text{closure} \\
\text{then} \ closure & := closure \cup V \\
\text{until } old = closure \\
\text{– If } T \subseteq \text{closure then } X \rightarrow T \text{ is entailed by } F
\end{align*}
\]
Example: Computation of Attribute Closure

**Problem:** Compute the attribute closure of $AB$ with respect to the set of FDs:

- $AB \rightarrow C$ (a)
- $A \rightarrow D$ (b)
- $D \rightarrow E$ (c)
- $AC \rightarrow B$ (d)

**Solution:**

Initially $\text{closure} = \{AB\}$

Using (a) $\text{closure} = \{ABC\}$

Using (b) $\text{closure} = \{ABCD\}$

Using (c) $\text{closure} = \{ABCD\}$

Example - Computing Attribute Closure

<table>
<thead>
<tr>
<th>$X$</th>
<th>$X_F^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>${A, D, E}$</td>
</tr>
<tr>
<td>$AB$</td>
<td>${A, B, C, D, E}$</td>
</tr>
</tbody>
</table>

(Hence $AB$ is a key)

Is $AB \rightarrow E$ entailed by $F$? Yes

Is $D \rightarrow C$ entailed by $F$? No

Result: $X_F^+$ allows us to determine FDs of the form $X \rightarrow Y$ entailed by $F$

Topics

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
- Normalization & Performance

Normal Forms

- Each normal form is a set of conditions on a schema that guarantees certain properties (relating to redundancy and update anomalies)
- First normal form (1NF):
  - each tuple = sequence of atomic values
  - same as the definition of relational model
- Second normal form (2NF):
  - has no practical or theoretical value – won’t discuss
- The two commonly used normal forms are third normal form (3NF) and Boyce-Codd normal form (BCNF)
BCNF

**Definition:** A relation schema $R$ is in BCNF if for every FD $X \rightarrow Y$ associated with $R$ either
- $Y \subseteq X$ (i.e., the FD is trivial) or
- $X$ is a superkey of $R$

**Example:** Person1($SSN$, $Name$, $Address$)
- The only FD is $SSN \rightarrow Name$, $Address$
- Since $SSN$ is a key, Person1 is in BCNF

(non) BCNF Examples

- **Person ($SSN$, $Name$, $Address$, $Hobby$)**
  - The FD $SSN \rightarrow Name$, $Address$ does not satisfy requirements of BCNF
  - since the key is ($SSN$, $Hobby$)
- **HasAccount ($AcctNum$, $ClientId$, $OfficeId$)**
  - The FD $AcctNum \rightarrow OfficeId$ does not satisfy BCNF requirements
  - since keys are ($ClientId$, $OfficeId$) and ($AcctNum$, $ClientId$), not $AcctNum$.

Redundancy & BCNF

- Suppose $R$ has a FD $A \rightarrow B$, and $A$ is not a superkey. If an instance has 2 rows with same value in $A$, they must also have same value in $B$ (⇒ redundancy, if the $A$-value repeats twice)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>stamps</td>
</tr>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>coins</td>
</tr>
</tbody>
</table>

- If $A$ is a superkey, there cannot be two rows with same value of $A$
  - Hence, BCNF eliminates redundancy

Third Normal Form

- A relational schema $R$ is in 3NF if for every FD $X \rightarrow Y$ associated with $R$ either:
  - $Y \subseteq X$ (i.e., the FD is trivial); or
  - $X$ is a superkey of $R$; or
  - Every $A \in Y$ is part of some key of $R$
- 3NF is weaker than BCNF
  - every scheme that is in BCNF is also in 3NF
### 3NF Example

- **HasAccount** \((AcctNum, ClientId, OfficeId)\)
  - \(ClientId, OfficeId \rightarrow AcctNum\)
  - OK since LHS contains a key
  - \(AcctNum \rightarrow OfficeId\)
  - OK since RHS is part of a key
- HasAccount is in 3NF but it might still contain redundant information due to \(AcctNum \rightarrow OfficeId\)
  - which is not allowed by BCNF

### 3NF (Non) Example

- **Person** \((SSN, Name, Address, Hobby)\)
  - \((SSN, Hobby)\) is the only key.
  - \(SSN \rightarrow Name\) violates 3NF conditions since \(Name\) is not part of a key and \(SSN\) is not a superkey

### Topics

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
  - Lossless
    - Dependency preserving
- Algorithms for BCNF and 3NF
- Normalization & Performance

### Lossless Decomposition

- **Goal:**
  - Eliminate redundancy by decomposing a relation into several relations in a higher normal form
- Decomposition must be lossless:
  - it must be possible to reconstruct the original relation from the relations in the decomposition
Decomposition: Definition

- Schema \( R = (R, F) \)
  - \( R \) is a set of attributes
  - \( F \) is a set of functional dependencies over \( R \)
  - Each key is described by a FD
- The decomposition of schema \( R \) is a collection of schemas \( R_i = (R_i, F_i) \) where
  - \( R = \bigcup_i R_i \) (no new, but all original attributes)
  - \( F_i \) is a set of functional dependencies involving only attributes of \( R_i \)
  - \( F \) entails \( F_i \) for all \( i \) (no new FDs)
- The decomposition of an instance, \( r \), of \( R \) is a set of relations \( r_i = \pi_{R_i}(r) \) for all \( i \)

Decomposition: Example

Schema \((R, F)\) where
\[ R = \{\text{SSN, Name, Address, Hobby}\} \]
\[ F = \{\text{SSN} \rightarrow \text{Name, Address}\} \]
can be decomposed into
\[ R_1 = \{\text{SSN, Name, Address}\} \]
\[ F_1 = \{\text{SSN} \rightarrow \text{Name, Address}\} \]
and
\[ R_2 = \{\text{SSN, Hobby}\} \]
\[ F_2 = \{\} \]

Lossless Schema Decomposition

- A decomposition should not lose information
- A decomposition \((R_1, \ldots, R_n)\) of a schema, \( R \), is lossless if every valid instance, \( r \), of \( R \) can be reconstructed from its components:
\[ r = r_1 \times r_2 \times \cdots \times r_n \]
- where each \( r_i = \pi_{R_i}(r) \)

Lossy Decomposition

The following is always the case (as long as join attributes don't have null values):
\[ r \subseteq r_1 \times r_2 \times \cdots \times r_n \]
But the following is not always true:
\[ r \supseteq r_1 \times r_2 \times \cdots \times r_n \]

Example:
\[
\begin{array}{|c|c|}
\hline
\text{SSN} & \text{Name} & \text{Address} \\
\hline
1111 & Joe & 1 Pine \\
2222 & Alice & 2 Oak \\
3333 & Alice & 3 Pine \\
\hline
\end{array}
\]
\[
\begin{array}{|c|c|}
\hline
\text{SSN} & \text{Name} \\
\hline
1111 & Joe \\
2222 & Alice \\
3333 & Alice \\
\hline
\end{array}
\]

The tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) are in the join, but not in the original.
Lossy Decompositions: What is Actually Lost?

• In the previous example, the tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) were gained, not lost!
  – Why do we say that the decomposition was lossy?

• What was lost is information:
  – That 2222 lives at 2 Oak: In the decomposition, 2222 can live at either 2 Oak or 3 Pine
  – That 3333 lives at 3 Pine: In the decomposition, 3333 can live at either 2 Oak or 3 Pine

Testing for Losslessness

• A (binary) decomposition of $R = (R, F)$ into $R_1 = (R_1, F_1)$ and $R_2 = (R_2, F_2)$ is lossless if and only if:
  – either the FD $\{ (R_1 \cap R_2) \rightarrow R_1 \}$ is in $F^+$
  – or the FD $\{ (R_1 \cap R_2) \rightarrow R_2 \}$ is in $F^+$

Example

Schema $(R, F)$ where

$R = \{ SSN, Name, Address, Hobby \}$

$F = \{ SSN \rightarrow Name, Address \}$

can be decomposed into

$R_1 = \{ SSN, Name, Address \}$

$F_1 = \{ SSN \rightarrow Name, Address \}$

and

$R_2 = \{ SSN, Hobby \}$

$F_2 = \{ \}$

Since $R_1 \cap R_2 = SSN$ and $SSN \rightarrow R_1$ the decomposition is lossless

Intuition Behind the Test for Losslessness

• Suppose $R_1 \cap R_2 \rightarrow R_2$. Then a row of $r_1$ can combine with exactly one row of $r_2$ in the natural join (since in $r_2$ a particular set of values for the attributes in $R_1 \cap R_2$ defines a unique row)
Topics

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
  - Lossless
  - Dependency preserving
- Algorithms for BCNF and 3NF
- Normalization & Performance

Is Lossless Decomposition the only Desirable Property?

- HasAccount (AcctNum, ClientId, OfficeId)
  \[ f_1: \text{AcctNum} \rightarrow \text{OfficeId} \]
  \[ f_2: \text{ClientId, OfficeId} \rightarrow \text{AcctNum} \]

- Decomposition:
  \[ R_1 = (\text{AcctNum, OfficeId}; \{ \text{AcctNum} \rightarrow \text{OfficeId} \}) \]
  \[ R_2 = (\text{AcctNum}, \text{ClientId}; \{\}) \]

- Decomposition is lossless:
  \[ R_1 \cap R_2 = \{\text{AcctNum}\} \]
  and \[ \text{AcctNum} \rightarrow \text{OfficeId} \]

- Both \[ R_1 \] and \[ R_2 \] are in BCNF
- But is something missing in the decomposition?
  \[ f_2 \]
  cannot be expressed on either \[ R_1 \] or \[ R_2 \]

Cost of Enforcing FDs

- Consider a decomposition of \[ R = (R, F) \] into \[ R_1 = (R_1, F_1) \] and \[ R_2 = (R_2, F_2) \]
  - An FD \[ X \rightarrow Y \] of \[ F^+ \] is in \[ F_1 \] iff \[ X \cup Y \subseteq R_1 \]
  - An FD, \[ f \in F^+ \] may be in neither \[ F_1 \], nor \[ F_2 \], nor even \[ (F_1 \cup F_2)^+ \]

- Checking that \[ f \] is true in \[ r_1 \] or \[ r_2 \] is (relatively) easy
- Checking \[ f \] in \[ r_1 \bowtie r_2 \] is harder – requires a join
- Ideally: want to check FDs locally, in \[ r_1 \] and \[ r_2 \], and have a guarantee that every \[ f \in F \] holds in \[ r_1 \bowtie r_2 \]

Dependency Preservation

- The decomposition is dependency preserving iff the sets \[ F \] and \[ F_1 \cup F_2 \] are equivalent: \[ F^+ = (F_1 \cup F_2)^+ \]
  - Then checking all FDs in \[ F \] as \[ r_1 \] and \[ r_2 \] are updated, can be done by checking \[ F_1 \] in \[ r_1 \] and \[ F_2 \] in \[ r_2 \]
- If \[ f \] is an FD in \[ F \] but \[ f \] is not in \[ F_1 \cup F_2 \] there are two possibilities:
  - \[ f \in (F_1 \cup F_2)^+ \]
    - If the constraints in \[ F_1 \] and \[ F_2 \] are maintained, \[ f \] will be maintained automatically.
  - \[ f \notin (F_1 \cup F_2)^+ \]
    - \[ f \] can be checked only by first taking the join of \[ r_1 \] and \[ r_2 \]. This is costly.
Example

Schema \((R, F)\) where

\[ R = \{\text{SSN}, \text{Name}, \text{Address}, \text{Hobby}\} \]
\[ F = \{\text{SSN} \rightarrow \text{Name}, \text{Address}\} \]

can be decomposed into

\[ R_1 = \{\text{SSN}, \text{Name}, \text{Address}\} \]
\[ F_1 = \{\text{SSN} \rightarrow \text{Name}, \text{Address}\} \]

and

\[ R_2 = \{\text{SSN}, \text{Hobby}\} \]
\[ F_2 = \{\} \]

Since \(F = F_1 \cup F_2\) the decomposition is dependency preserving.

Example

• Schema: \((\text{ABC}; F), F = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}\)

• Decomposition:
  – \((AC, F_1), F_1 = \{A \rightarrow C\}\)
    • Note: \(A \rightarrow C \notin F\), but in \(F^+\)
  – \((BC, F_2), F_2 = \{B \rightarrow C, C \rightarrow B\}\)

• \(A \rightarrow B \notin (F_1 \cup F_2)\), but \(A \rightarrow B \in (F_1 \cup F_2)^+\).
  – So \(F^+ = (F_1 \cup F_2)^+\) and thus the decompositions is still dependency preserving.

Example

• HasAccount \((\text{AccntNum}, \text{ClientId}, \text{OfficeId})\)
  \[ f_1: \text{AccntNum} \rightarrow \text{OfficeId} \]
  \[ f_2: \text{ClientId, OfficeId} \rightarrow \text{AccntNum} \]

• Decomposition:
  \[ R_1 = (\text{AccntNum, OfficeId}; \{\text{AccntNum} \rightarrow \text{OfficeId}\}) \]
  \[ R_2 = (\text{AccntNum, ClientId}; \{\}) \]

• Decomposition is lossless:
  \[ R_1 \cap R_2 = \{\text{AccntNum}\} \text{ and AccntNum} \rightarrow \text{OfficeId} \]

• In BCNF
  • Not dependency preserving: \(f_2 \notin (F_1 \cup F_2)^+\)

  • HasAccount does not have BCNF decompositions that are both lossless and dependency preserving! (Check, e.g. by enumeration)

  • Hence: BCNF+lossless+dependency preserving decompositions are not always achievable!

Topics

• Limitations of ER modeling
• Functional Dependencies
• Normal Forms
• Decompositions
  → Algorithms for BCNF and 3NF
• Normalization & Performance
Third Normal Form

- **Compromise** – Not all redundancy removed, but dependency preserving decompositions are always possible (and, of course, lossless)
- **3NF decomposition** is based on a **minimal cover**
  - Another name for minimal cover is **canonical cover**

Minimal Cover

- A **minimal cover** of a set of dependencies, $F$, is a set of dependencies, $U$, such that:
  - $U$ is equivalent to $F$ ($F^+ = U^+$)
  - All FDs in $U$ have the form $X \rightarrow A$ where $A$ is a single attribute
  - It is not possible to make $U$ smaller (while preserving equivalence) by
    - Deleting an FD
    - Deleting an attribute from an FD (either from LHS or RHS)
  - FDs and attributes that can be deleted in this way are called **redundant**

Computing Minimal Cover

- **Example**: $F = \{ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, 
  BGH \rightarrow L, L \rightarrow AD, E \rightarrow L, BH \rightarrow E\}$
- **step 1**: Make RHS of each FD into a single attribute
  - **Algorithm**: Use the decomposition inference rule for FDs
    - Example: $L \rightarrow AD$ replaced by $L \rightarrow A, L \rightarrow D$; $ABH \rightarrow CK$ by $ABH \rightarrow C, ABH \rightarrow K$
- **step 2**: Eliminate redundant attributes from LHS.
  - **Algorithm**: If FD $XB \rightarrow A \in F$ (where $B$ is a single attribute) and $X \rightarrow A$ is entailed by $F$, then $B$ was unnecessary
  - Example: Can an attribute be deleted from $ABH \rightarrow C$?
    - Compute $ABF$, and $AIF$ and $BIF$
    - Since $C \in (BIF)$, therefore $BH \rightarrow C$ is entailed by $F$ and $A$ is redundant in $ABH \rightarrow C$.
- **step 3**: Delete redundant FDs from $F$
  - **Algorithm**: If $F - \{f\}$ entails $f$, then $f$ is redundant
    - If $f$ is $X \rightarrow A$ then check if $A \in X^+$
    - Example: $BGH \rightarrow L$ is entailed by $E \rightarrow L, BH \rightarrow E$, so it is redundant
- **Note**:
  - The order of steps 2 and 3 cannot be interchanged! See the textbook for a counterexample
  - A minimal cover might not be unique!
Synthesizing a 3NF Schema
Starting with a schema \( R = (R, F) \)

- **step 1**: Compute a minimal cover, \( U \), of \( F \). The decomposition is based on \( U \), but since \( U^+ = F^+ \) the same functional dependencies will hold
  - A minimal cover for \( F = \{ ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow AD, E \rightarrow L, BH \rightarrow E \} \) is
  \[ U = \{ BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, L \rightarrow A, E \rightarrow L \} \]

Synthesizing a 3NF Schema (cont’d)

- **step 2**: Partition \( U \) into sets \( U_1, U_2, \ldots, U_n \) such that the LHS of all elements of \( U_i \) are the same
  - \( U_1 = \{ BH \rightarrow C, BH \rightarrow K \} \), \( U_2 = \{ A \rightarrow D \} \), \( U_3 = \{ C \rightarrow E \} \), \( U_4 = \{ L \rightarrow A \} \), \( U_5 = \{ E \rightarrow L \} \)

Synthesizing a 3NF schema (cont’d)

- **step 3**: For each \( U_i \) form schema \( R_i = (R_i, U_i) \), where \( R_i \) is the set of all attributes mentioned in \( U_i \)
  - Each FD of \( U \) will be in some \( R_i \). Hence the decomposition is dependency preserving
  - \( R_1 = (BHCK; BH \rightarrow C, BH \rightarrow K) \), \( R_2 = (AD; A \rightarrow D) \), \( R_3 = (CE; C \rightarrow E) \), \( R_4 = (AL; L \rightarrow A) \), \( R_5 = (EL; E \rightarrow L) \)

- **step 4**: If no \( R_i \) is a superkey of \( R \), add schema \( R_0 = (R_0, \{\}) \) where \( R_0 \) is a key of \( R \)
  - \( R_0 = (BGH; \{\}) \)
    - \( R_0 \) might be needed when not all attributes are necessarily contained in \( R_1 \cup R_2 \cup \cdots \cup R_n \)
    - A missing attribute, \( A \), must be part of all keys
      (since it’s not in any FD of \( U \), deriving a key constraint from \( U \) involves the augmentation axiom)
    - \( R_0 \) might be needed even if all attributes are accounted for in \( R_1 \cup R_2 \cup \cdots \cup R_n \)
      - Example: \( (ABCD; \{A \rightarrow B, C \rightarrow D\}) \)
      - Step 3 decomposition: \( R_1 = (AB; \{A \rightarrow B\}) \), \( R_2 = (CD; \{C \rightarrow D\}) \)
      - Lossy! Need to add \( (AC; \{\}) \), for losslessness
    - Step 4 guarantees lossless decomposition.
Topics

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
  - 3 NF Synthesis Algorithm
  - BCNF Decomposition Algorithm
- Normalization & Performance

BCNF Decomposition Algorithm

- You can skip this topic (Section 6.7 in the book)
- Briefly:
  - The BCNF decomposition algorithm works by using an FD that violates BCNF form and uses this FD to decompose the schema
  - The FDs attached to a decomposed schema include all FDs involving attributes of the decomposed schema
    - This set must include any FDs entailed in the original set of FDs
  - The algorithm given in the book has exponential complexity
  - Alternate (more complex) algorithm exists with polynomial complexity

BCNF Design Strategy

If we use the 3NF Synthesis Algorithm

- The resulting decomposition, \( R_0, R_1, \ldots, R_n \), is
  - Dependency preserving (since every FD in \( U \) is a FD of some schema)
  - Lossless (although this is not obvious)
  - In 3NF (although this is not obvious)
- Strategy for decomposing a relation
  - Use 3NF decomposition first to get lossless, dependency preserving decomposition
  - If any resulting schema is not in BCNF, split it using the BCNF algorithm (but this may yield a non-dependency preserving result)

Topics

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
  - Normalization & Performance
**Normalization Drawbacks**

- By limiting redundancy, normalization helps maintain consistency and saves space.
- But performance of querying can suffer because related information that could be stored in a single relation is now distributed among several.
- **Example**: A join is required to get the names and hobbies of all persons.

```sql
SELECT P.Name, H.Hobby
FROM Person P, Hobby H
WHERE P.SSN = H.SSN
```

**Denormalization**

- **Tradeoff**: Judiciously introduce redundancy to improve performance of certain queries.
- **Example**: In our Student registration schema
  - to find the names of students taking a given class in a particular semester
  - a join is required between Student and Transcript table:

```sql
SELECT S.Name, T.Grade
FROM Student S, Transcript T
WHERE S.Id = T.StudId AND
  T.CrsCode = 'CS305' AND T.Semester = 'S2004'
```

**Denormalization (cont’d)**

- Add attribute Name to Transcript
  
```sql
SELECT T.Name, T.Grade
FROM Transcript T
WHERE T.CrsCode = 'CS305' AND T.Semester = 'S2002'
```
  
  Join is avoided
- If queries are asked more frequently than Transcript is modified, added redundancy might improve average performance.
- But, Transcript1 is no longer in BCNF
  - since key is (StudId, CrsCode, Semester) and StudId → Name
- Updates to Student.Name must keep the database consistent
  - These updates are also slowed down

**Topics**

- Limitations of ER modeling
- Functional Dependencies
- Normal Forms
- Decompositions
- Algorithms for BCNF and 3NF
- Normalization & Performance
- Summary
Summary

• In the beginning we noted some of the problems with ER modeling:
  – Provides a set of guidelines, does not result in a unique database schema
  – Does not provide a way of evaluating alternative schemas
• Normalization theory was introduced to solve these problems
  – (How) did we solve the problems?

Summary (cont’d)

• The key concepts were functional dependencies, normal forms and decompositions
• FDs must be identified during the conceptual schema design
• Using FDs, we can determine if a schema has redundancy due to not being normalized
  – We do not need an instance of the schema to determine redundancy
  – We can reason based on FDs to determine properties that must hold on all valid instances of the schema
• 3 NF and BCNF are two desirable forms of the schema

Summary (cont’d)

• The solution to the redundancy problem is decomposition to a desired normal form
• We identified desirable properties of decompositions
  – Losslessness, dependency preservation
  – For an arbitrary schema, it is not guaranteed that a BCNF decomposition will have both these properties
  – Always possible to have a 3 NF decomposition which is both lossless and preserves dependency

Summary (cont’d)

• Algorithms exist to normalize a schema to 3 NF or to BCNF
  – Studied the 3 NF synthesis algorithm
  – A unique database schema is not guaranteed
  – Not that big an issue, since any schema produced by the algorithm will be in 3 NF, lossless, dependency preserving
• Normalization can reduce performance of queries
  – Denormalize? Be careful
**Summary (cont’d)**

- In some relational schemas, even a schema in BCNF would suffer from the problem of repetition of information
- This sort of redundancy can be eliminated by higher forms of normalization: 4NF, 5NF
  - Using the concept of multivalued dependencies (for 4NF), join dependencies (5NF)

**Fourth Normal Form**

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneN</th>
<th>ChildSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111</td>
<td>123-4444</td>
<td>222222</td>
</tr>
<tr>
<td>11111</td>
<td>123-4444</td>
<td>333333</td>
</tr>
<tr>
<td>11111</td>
<td>321-5555</td>
<td>222222</td>
</tr>
<tr>
<td>11111</td>
<td>321-5555</td>
<td>333333</td>
</tr>
<tr>
<td>22222</td>
<td>987-6666</td>
<td>444444</td>
</tr>
<tr>
<td>22222</td>
<td>777-7777</td>
<td>444444</td>
</tr>
<tr>
<td>22222</td>
<td>987-6666</td>
<td>555555</td>
</tr>
<tr>
<td>22222</td>
<td>777-7777</td>
<td>555555</td>
</tr>
</tbody>
</table>

- Relation has redundant data
- Yet it is in BCNF (since there are no non-trivial FDs)
- Redundancy is due to set valued attributes (in the E-R sense), not because of the FDs