Last Time

• Introduced Third Normal Form
  – A weakened version of BCNF that preserves more functional dependencies
  – Allows non-trivial dependencies $\alpha \rightarrow \beta$ if attributes in $(\beta - \alpha)$ also appear in candidate keys
  – Allows more redundant data than BCNF

• Began discussing functional dependency theory
  – Rules of inference for functional dependencies
  – The closure of a set of functional dependencies
  – The closure of a particular attribute-set
Canonical Cover

• Given a relation schema, and a set of functional dependencies $F$
• Database needs to enforce $F$ on all relations
  – Invalid changes should be rolled back
• $F$ may contain many functional dependencies
  – Dependencies might even imply each other
• Want a minimal version of $F$, that still represents all constraints imposed by $F$
  – Should be more efficient to enforce minimal version
Extraneous Attributes

• Given a set $F$ of functional dependencies
  – An attribute in a functional dependency is *extraneous* if it can be removed from $F$ without affecting the closure of $F$

• Formally: given $F$, and $\alpha \rightarrow \beta$
  – If $A \in \alpha$, and $F$ logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$, then $A$ is extraneous
  – If $A \in \beta$, and $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies $F$, then $A$ is extraneous
Testing Extraneous Attributes

- Given relation schema $R$, and a set $F$ of functional dependencies that hold on $R$
- Attribute $A$ in $\alpha \rightarrow \beta$
- If $A \in \beta$, then take new set
  - $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - See if $\alpha \rightarrow A$ can be inferred from $F'$
  - Compute $\alpha^+$ under $F'$
  - If $\alpha^+$ includes $A$ then $A$ is extraneous in $\beta$
Testing Extraneous Attributes (2)

- Given relation schema $R$, and a set $F$ of functional dependencies that hold on $R$
- Attribute $A$ in $\alpha \rightarrow \beta$
- If $A \in \alpha$, then let $\gamma = \alpha - \{A\}$
  - See if $\gamma \rightarrow \beta$ can be inferred from $F$
  - Compute $\gamma^+$ under $F$
  - If $\gamma^+ \supseteq \beta$ then $A$ is extraneous in $\alpha$
Canonical Cover

• A canonical cover $F_c$ for $F$ is a set of functional dependencies such that:
  – $F$ logically implies all dependencies in $F_c$
  – $F_c$ logically implies all dependencies in $F$
  – No functional dependency in $F_c$ contains an extraneous attribute
  – Left side of all functional dependencies in $F_c$ are unique
    • There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in $F_c$ such that $\alpha_1 = \alpha_2$
Computing Canonical Cover

• A simple way to compute the canonical cover of $F$

$$F_c = F$$

**repeat**

apply union rule to replace dependencies in $F_c$ of form

$$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

find a functional dependency $\alpha \rightarrow \beta$ in $F_c$ with an extraneous attribute

/* Use $F_c$ for extraneous attribute test, not $F$! */

if an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

**until** $F_c$ stops changing
Canonical Cover Example

- Functional dependencies $F$ on schema $(A, B, C)$
  - $F = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$
  - Find $F_c$
- Apply union rule to $A \rightarrow BC$ and $A \rightarrow B$
  - Left with: $\{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$
- $A$ is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ is logically implied by $F$ (obvious)
  - Left with: $\{ A \rightarrow BC, B \rightarrow C \}$
- $C$ is extraneous in $A \rightarrow BC$
  - Logically implied by $A \rightarrow B, B \rightarrow C$
- $F_c = \{ A \rightarrow B, B \rightarrow C \}$
Canonical Covers

• A set of functional dependencies can have multiple canonical covers

• Example:
  – \( F = \{ A \rightarrow BC, B \rightarrow AC, C \rightarrow AB \} \)
  – Has several canonical covers:
    • \( F_c = \{ A \rightarrow B, B \rightarrow C, C \rightarrow A \} \)
    • \( F_c = \{ A \rightarrow B, B \rightarrow AC, C \rightarrow B \} \)
    • \( F_c = \{ A \rightarrow C, C \rightarrow B, B \rightarrow A \} \)
    • \( F_c = \{ A \rightarrow C, B \rightarrow C, C \rightarrow AB \} \)
    • \( F_c = \{ A \rightarrow BC, B \rightarrow A, C \rightarrow A \} \)
Lossy Decompositions

• Some schema decompositions lose information
• Example:

  \texttt{employee(emp\_id, emp\_name, phone, title, salary, start\_date)}
  
  – Decomposed into:

  \texttt{emp\_ids(emp\_id, emp\_name)}
  \texttt{emp\_details(emp\_name, phone, title, salary, start\_date)}

• Problem:
  
  – \texttt{emp\_name} doesn’t uniquely identify employees
  – This is a lossy decomposition
Lossless Decompositions

• Given:
  – Relation schema \( R \), relation \( r(R) \)
  – Set of functional dependencies \( F \)

• Let \( R_1 \) and \( R_2 \) be a decomposition of \( R \)

• The decomposition is lossless if, for all legal instances of \( r \):
  \[
  \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r
  \]

• A simple definition…
Lossless Decompositions (2)

- Can define with functional dependencies:
  - $R_1$ and $R_2$ form a lossless decomposition of $R$ if at least one of these dependencies is in $F^+$:
    \[ R_1 \cap R_2 \rightarrow R_1 \]
    \[ R_1 \cap R_2 \rightarrow R_2 \]

- $R_1 \cap R_2$ forms a superkey of $R_1$ and/or $R_2$
  - Can test for superkeys using attribute closure
Employee Database Example

• For employee example:
  
  employee(emp_id, emp_name, phone, title, salary, start_date)
  
  – Decomposed into:
    
    emp_ids(emp_id, emp_name)
    
    emp_details(emp_name, phone, title, salary, start_date)

• emp_name is not a superkey of emp_ids or emp_details
BCNF Decompositions

• If $R$ is a schema not in BCNF:
  – There is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that $\alpha$ is not a superkey for $R$
  – For simplicity, also require that $\alpha \cap \beta = \emptyset$
• Replace $R$ with two schemas:
  $R_1 = (\alpha \cup \beta)$
  $R_2 = (R - \beta)$
• BCNF decomposition is lossless
  – $R_1 \cap R_2 = \alpha$
  – $\alpha$ is a superkey of $R_1$
  – $\alpha$ also appears in $R_2$
Dependency Preservation

• Some schema decompositions are not dependency-preserving
  – Functional dependencies that span multiple relation schemas are hard to enforce
  – e.g. BCNF may require decomposition of a schema for one dependency, and make it hard to enforce another dependency

• Can test for dependency preservation using functional dependency theory
Dependency Preservation (2)

• Given:
  – A set $F$ of functional dependencies on a schema $R$
  – $R_1, R_2, \ldots, R_n$ are a decomposition of $R$

• The restriction of $F$ to $R_i$ is the set $F_i$ of functional dependencies in $F^+$ that only has attributes in $R_i$
  – Each $F_i$ contains functional dependencies that can be checked efficiently, using $R_i$

• Find all functional dependencies that can be checked efficiently
  – $F' = F_1 \cup F_2 \cup \ldots \cup F_n$
  – If $F'^+ = F^+$ then the decomposition is dependency-preserving
Third Normal Form Schemas

• Can generate a 3NF schema from a set of functional dependencies $F$

• Called the 3NF synthesis algorithm
  – Instead of decomposing an initial schema, generates schemas from a set of dependencies

• Given a set $F$ of functional dependencies
  – Uses the canonical cover $F_c$
  – Ensures that resulting schemas are dependency-preserving
3NF Synthesis Algorithm

• Inputs: set of functional dependences $F$, on a schema $R$

let $F_c$ be a canonical cover for $F$;
i := 0;
for each functional dependency $\alpha \rightarrow \beta$ in $F_c$ do
  if none of the schemas $R_j, j = 1, 2, \ldots, i$ contains $(\alpha \cup \beta)$ then
    i := i + 1;
    $R_i := (\alpha \cup \beta)$
  end if
end for

if no schema $R_j, j = 1, 2, \ldots, i$ contains a candidate key for $R$ then
  i := i + 1;
  $R_i :=$ any candidate key for $R$
end if
return $(R_1, R_2, \ldots, R_i)$
BCNF vs. 3NF vs. SQL

- **Boyce-Codd Normal Form:**
  - Eliminates more redundant information
  - Harder to enforce some functional dependencies
  - Overall, very desirable normal form

- **Third Normal Form:**
  - All functional dependencies are “easy” to enforce
  - Allows redundant information, which must be kept synchronized

- **SQL constraints:**
  - Only key constraints are fast and easy to enforce!
  - Only easy to enforce functional dependencies \( \alpha \rightarrow \beta \) if \( \alpha \) is a key on some table!
  - Other functional dependencies (even the “easy” ones) may require more expensive constraints, e.g. **CHECK** constraints
BCNF vs. 3NF vs. SQL (2)

- For SQL databases with materialized views:
  - Can decompose schema into BCNF
  - For dependencies $\alpha \rightarrow \beta$ not preserved in the decomposition, can create a materialized view joining all relations in the dependency
  - Enforce unique($\alpha$) constraint on materialized view

- Impacts both space and performance, but it works…
Next Time

- Functional dependencies are insufficient to represent all kinds of dependencies
- Next time:
  - Multivalued dependencies
  - Fourth Normal Form