Normal forms specify “good schema” patterns

First normal form (1NF):
- All attributes must be atomic
- Easy in relational model, harder/less desirable in SQL

Boyce-Codd normal form (BCNF):
- Eliminates redundancy using functional dependencies
- Given a relation $R$ and a set of dependencies $F$
- For all functional dependencies $\alpha \rightarrow \beta$ in $F^+$, where $\alpha \cup \beta \subseteq R$, at least one of these conditions must hold:
  - $\alpha \rightarrow \beta$ is a trivial dependency
  - $\alpha$ is a superkey for $R$
Can convert a schema into BCNF

If \( R \) is a schema not in BCNF:
- There is at least one nontrivial functional dependency \( \alpha \rightarrow \beta \in F^+ \) such that \( \alpha \) is not a superkey for \( R \)

Replace \( R \) with two schemas:
- \( (\alpha \cup \beta) \)
- \( (R - (\beta - \alpha)) \)

May need to repeat this decomposition process until all schemas are in BCNF
Functional Dependency Theory

- Important to be able to reason about functional dependencies!
- Main question:
  - What functional dependencies are implied by a set $F$ of functional dependencies?
- Other useful questions:
  - Which attributes are functionally determined by a particular attribute-set?
  - What *minimal* set of functional dependencies must actually be enforced in a database?
  - Is a particular schema decomposition lossless?
  - Does a decomposition preserve dependencies?
Rules of Inference

- Given a set $F$ of functional dependencies
  - Actual dependencies listed in $F$ may be insufficient for normalizing a schema
  - Must consider all dependencies logically implied by $F$

- For a relation schema $R$
  - A functional dependency $f$ on $R$ is logically implied by $F$ on $R$ if every relation instance $r(R)$ that satisfies $F$ also satisfies $f$

- Example:
  - Dependencies: $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$
  - Logically implies: $A \rightarrow H$, $CG \rightarrow HI$, $AG \rightarrow I$
Rules of Inference (2)

- **Axioms** are rules of inference for dependencies
- This group is called Armstrong’s axioms
- Greek letters $\alpha$, $\beta$, $\gamma$, … represent attribute sets
- **Reflexivity rule:**
  
  If $\alpha$ is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

- **Augmentation rule:**
  
  If $\alpha \rightarrow \beta$ holds, and $\gamma$ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

- **Transitivity rule:**
  
  If $\alpha \rightarrow \beta$ holds, and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.
Computing Closure of $F$

Can use Armstrong’s axioms to compute $F^+$ from $F$

- $F$ is a set of functional dependencies

\[
F^+ = F
\]

repeat

for each functional dependency $f$ in $F^+$

- apply reflexivity and augmentation rules to $f$
- add resulting functional dependencies to $F^+$

for each pair of functional dependencies $f_1, f_2$ in $F^+$

- if $f_1$ and $f_2$ can be combined using transitivity
  - add resulting functional dependency to $F^+$

until $F^+$ stops changing
Armstrong’s Axioms

- **Axioms are sound**
  - They don’t generate any incorrect functional dependencies

- **Axioms are complete**
  - Given a set of functional dependencies $F$, repeated application generates all $F^+$

- $F^+$ could be very large
  - LHS and RHS of a dependency are subsets of $R$
  - A set of size $n$ has $2^n$ subsets
  - $2^n \times 2^n = 2^{2n}$ possible functional dependencies in $R$!
More Rules of Inference

- Additional rules can be proven from Armstrong’s axioms
  - These make it easier to generate $F^+$

- Union rule:
  If $\alpha \rightarrow \beta$ holds, and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.

- Decomposition rule:
  If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

- Pseudotransitivity rule:
  If $\alpha \rightarrow \beta$ holds, and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
Attribute-Set Closure

- **How to tell if an attribute-set $\alpha$ is a superkey?**
  - If $\alpha \rightarrow R$ then $\alpha$ is a superkey.
  - What attributes are functionally determined by an attribute-set $\alpha$?

- **Given:**
  - Attribute-set $\alpha$
  - Set of functional dependencies $F$
  - The set of all attributes functionally determined by $\alpha$ under $F$ is called the closure of $\alpha$ under $F$
  - Written as $\alpha^+$
Attribute-Set Closure (2)

- It’s easy to compute the closure of attribute-set $\alpha$!
  - Algorithm is very simple

- Inputs:
  - attribute-set $\alpha$
  - set of functional dependencies $F$

\[
\alpha^+ = \alpha
\]

repeat
  for each functional dependency $\beta \rightarrow \gamma$ in $F$
    if $\beta \subseteq \alpha^+$ then
      $\alpha^+ = \alpha^+ \cup \gamma$
  until $\alpha^+$ stops changing
Attribute-Set Closure (3)

- Can easily test if $\alpha$ is a superkey
  - Compute $\alpha^+$
  - If $R \subseteq \alpha^+$ then $\alpha$ is a superkey of $R$

- Can also use to identify functional dependencies
  - $\alpha \rightarrow \beta$ holds if $\beta \subseteq \alpha^+$
    - Find closure of $\alpha$ under $F$; if it contains $\beta$ then $\alpha \rightarrow \beta$ holds!
  - Can compute $F^+$ with attribute-set closure too:
    - For each $\gamma \subseteq R$, find closure $\gamma^+$ under $F$
      - We know that $\gamma \rightarrow \gamma^+$
    - For each subset $S \subseteq \gamma^+$, add functional dependency $\gamma \rightarrow S$
Attribute-Set Closure Example

- Relation schema \( R(A, B, C, G, H, I) \)
  - Dependencies:
    \( A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \)
- Is \( AG \) a superkey of \( R \)?
- Compute \((AG)^+\)
  - Start with \( \alpha^+ = AG \)
  - \( A \rightarrow B, A \rightarrow C \) cause \( \alpha^+ = ABCG \)
  - \( CG \rightarrow H, CG \rightarrow I \) cause \( \alpha^+ = ABCGHI \)
- \( AG \) is a superkey of \( R \)!
Attribute-Set Closure Example (2)

  - Dependencies: $A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H$

- Is $AG$ a candidate key of $R$?
  - A candidate key is a minimal superkey
  - Compute attribute-set closure of all proper subsets of superkey; if we get $R$ then it’s not a candidate key

- Compute the attribute-set closures under $F$
  - $A^+ = ABCH$
  - $G^+ = G$

- $AG$ is indeed a candidate key!
BCNF Revisited

BCNF algorithm states, if \( R_i \) 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 \( R_i \).

Two points:
- \( \alpha \rightarrow \beta \in F^+ \), not just in \( F \)
- For \( R_i \), only care about func. deps. where \( \alpha \cup \beta \in R_i \)

How do we tell if \( R_i \) is not in BCNF?
- Can use attribute-set closure under \( F \) to find if there is a dependency in \( F^+ \) that affects \( R_i \)
- For each proper subset \( \alpha \subset R_i \), compute \( \alpha^+ \) under \( F \)
- If \( \alpha^+ \) doesn’t contain \( R_i \), but \( \alpha^+ \) does contain any attributes in \( R_i - \alpha \), then \( R_i \) is not in BCNF
BCNF Revisited (2)

- If $\alpha^+$ doesn’t contain $R_i$, but $\alpha^+$ does contain any attributes in $R_i - \alpha$, then $R_i$ is not in BCNF.

- If $\alpha^+$ doesn’t contain $R_i$, what do we know about $\alpha$ with respect to $R_i$?
  - $\alpha$ is not a superkey of $R_i$.

- If $\alpha^+$ contains attributes in $R_i - \alpha$:
  - Let $\beta = R_i \cap (\alpha^+ - \alpha)$
  - We know there is some non-trivial functional dependency $\alpha \rightarrow \beta$ that holds on $R_i$.
  - Since $\alpha \rightarrow \beta$ holds on $R_i$, but $\alpha$ is not a candidate key of $R_i$, we know that $R_i$ cannot be in BCNF.
BCNF Example

- Start with schema $R(A, B, C, D, E)$, and $F = \{ A \rightarrow B, BC \rightarrow D \}$
- Is $R$ in BCNF?
  - Obviously not.
  - Using $A \rightarrow B$, decompose into $R_1(A, B)$ and $R_2(A, C, D, E)$
- Are we done?
  - Pseudotransitivity rule says that if $\alpha \rightarrow \beta$ and $\gamma \beta \rightarrow \delta$, then $\alpha \gamma \rightarrow \delta$
  - $AC \rightarrow D$ also holds on $R_2$, so $R_2$ is not in BCNF!
  - Or, compute $\{AC\}^+ = ABCD$. Again, $R_2$ is not in BCNF.
Enforcing database constraints can easily become very expensive

- Especially CHECK constraints!

Best to define database schema such that constraint enforcement is efficient

Ideally, enforcing a functional dependency involves only one relation

- Then, can specify a key constraint instead of a multi-table CHECK constraint!
Example: Personal Bankers

- Bank sets a requirement on employees:
  - Each employee can work at only one branch
  - $emp_{id} \rightarrow branch\_name$

- Bank wants to give customers a personal banker at each branch
  - At each branch, a customer has only one personal banker
  - (A customer could have personal bankers at multiple branches.)
  - $cust\_id, branch\_name \rightarrow emp_{id}$
E-R diagram:

```
works_in

branch
branch_name
branch_city
assets

cust_banker_branch
cust_id
cust_name
type

employee
emp_id
emp_name

customer

works_in(emp_id, branch_name)
cust_banker_branch(cust_id, branch_name, emp_id, type)
```
Personal Bankers (2)

- **Schemas:**
  - \( \text{works\_in}(\text{emp\_id}, \text{branch\_name}) \)
  - \( \text{cust\_banker\_branch}(\text{cust\_id}, \text{branch\_name}, \text{emp\_id}, \text{type}) \)

- **Is this schema in BCNF?**
  - \( \text{emp\_id} \rightarrow \text{branch\_name} \)
  - \( \text{cust\_banker\_branch} \) isn’t in BCNF
    - \( \text{emp\_id} \) isn’t a candidate key on \( \text{cust\_banker\_branch} \)
    - \( \text{cust\_banker\_branch} \) repeats \( \text{branch\_name} \) unnecessarily, since \( \text{emp\_id} \rightarrow \text{branch\_name} \)

- **Decompose into two BCNF schemas:**
  - \( \text{works\_in} \) already has \( (\text{emp\_id}, \text{branch\_name}) \) \( (\alpha \cup \beta) \)
  - Create \( \text{cust\_banker}(\text{cust\_id}, \text{emp\_id}, \text{type}) \) \( (R - (\beta - \alpha)) \)
New BCNF schemas:

- `works_in(emp_id, branch_name)`
- `cust_banker(cust_id, emp_id, type)`

A customer can have one personal banker at each branch, so both `cust_id` and `emp_id` must be in the primary key.

Any problems with this new BCNF version?

- Now we can’t easily constrain that each customer has only one personal banker at each branch!
- Could still create a complicated **CHECK** constraint involving multiple tables…
Preserving Dependencies

- The BCNF decomposition doesn’t preserve this dependency:
  - cust_id, branch_name → emp_id
  - Can’t enforce this dependency within a single table

- In general, BCNF decompositions are not dependency-preserving
  - Some functional dependencies are not enforceable within a single table
  - Can’t enforce them with a simple key constraint, so they are more expensive

- Solution: Third Normal Form
Third Normal Form

- Slightly weaker than Boyce-Codd normal form
  - Preserves more functional dependencies
  - Also allows more repeated information!

- Given:
  - Relation schema $R$
  - Set of functional dependencies $F$

- $R$ is in 3NF with respect to $F$ if:
  - For all functional dependencies $\alpha \rightarrow \beta$ in $F^+$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial dependency
    - $\alpha$ is a superkey for $R$
    - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$
Third Normal Form (2)

- New condition:
  - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$

- A general constraint:
  - Doesn’t require a single candidate key to contain all attributes in $\beta - \alpha$
  - Just requires that each attribute in $\beta - \alpha$ appears in some candidate key in $R$
  - …possibly even different candidate keys!
Our non-BCNF personal banker schemas again:

- \( \text{works\_in}(\text{emp\_id}, \text{branch\_name}) \)
- \( \text{cust\_banker\_branch}(\text{cust\_id}, \text{branch\_name}, \text{emp\_id}, \text{type}) \)

Is this schema in 3NF?

- \( \text{emp\_id} \rightarrow \text{branch\_name} \)
- \( \text{cust\_id, branch\_name} \rightarrow \text{emp\_id} \)

\( \text{works\_in} \) is in 3NF (\( \text{emp\_id} \) is the primary key)

What about \( \text{cust\_banker\_branch} \) ?

- Both dependencies hold on \( \text{cust\_banker\_branch} \)
  - \( \text{emp\_id} \rightarrow \text{branch\_name} \), but \( \text{emp\_id} \) isn’t the primary key
  - \( \text{cust\_id, branch\_name} \rightarrow \text{emp\_id} \); is \( \text{emp\_id} \) part of any candidate key on \( \text{cust\_banker\_branch} \)?
Look carefully at the functional dependencies:

- Primary key of `cust_banker_branch` is `(cust_id, branch_name)`
  - `{ cust_id, branch_name } → cust_banker_branch (all attributes) (constraint arises from the E-R diagram & schema translation)
  - (Also specified this constraint: `cust_id, branch_name → emp_id`)
- We also know that `emp_id → branch_name`
- Pseudotransitivity rule: if $\alpha \rightarrow \beta$ and $\gamma \beta \rightarrow \delta$, then $\alpha \gamma \rightarrow \delta$
  - `{ emp_id } → { branch_name }
  - `{ cust_id, branch_name } → cust_banker_branch
  - Therefore, `{ emp_id, cust_id } → cust_banker_branch` also holds!
- `(cust_id, emp_id)` is a candidate key of `cust_banker_branch`

So `cust_banker_branch` is in fact in 3NF

(And we need to enforce this second candidate key too...)
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
- $F$ could contain a lot of functional dependencies
  - Dependencies might even logically 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
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 $F$
- Can’t infer any functional dependency in $F_c$ from other dependencies in $F_c$
- 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 $\alpha_1 = \alpha_2$
Extraneous Attributes

- Given a set of functional dependencies $F$
  - An attribute in a functional dependency is extraneous if it can be removed from $F$ without affecting 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
    - i.e. generate a new set of functional dependencies $F'$ by replacing $\alpha \rightarrow \beta$ with $\alpha \rightarrow (\beta - A)$$
    - See if $F'$ logically implies $F$
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 \( A \in \alpha \) (i.e. \( A \) is on left side of the dependency), then let \( \gamma = \alpha - \{A\} \)
  - See if \( \gamma \rightarrow \beta \) can be inferred from \( F \)
  - Compute \( \gamma^+ \) under \( F \)
  - If \( \beta \supseteq \gamma^+ \), then \( A \) is extraneous in \( \alpha \)
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 \( A \in \beta \) (on right side of the dependency), then try the altered set \( F' \)
  - \( F' = (F - \{ \alpha \rightarrow \beta \}) \cup \{ \alpha \rightarrow (\beta - A) \} \)
  - See if \( \alpha \rightarrow A \) can be inferred from \( F' \)
  - Compute \( \alpha^+ \) under \( F' \)
  - If \( \alpha^+ \) includes \( A \), then \( A \) is extraneous in \( \beta \)
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
    $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ 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 the 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 $A \rightarrow B, B \rightarrow C$

- $F_c = \{ A \rightarrow B, B \rightarrow C \}$
Another Example

- Functional dependencies $F$ on schema $(A, B, C, D)$
  - $F = \{ A \rightarrow B, BC \rightarrow D, AC \rightarrow D \}$
  - Find $F_c$

- In this case, it may look like $F_c = F$...

- However, can infer $AC \rightarrow D$ from $A \rightarrow B, BC \rightarrow D$ (pseudotransitivity), so $AC \rightarrow D$ is extraneous in $F$
  - Therefore, $F_c = \{ A \rightarrow B, BC \rightarrow D \}$

- Alternately, can argue that $D$ is extraneous in $AC \rightarrow D$
  - With $F' = \{ A \rightarrow B, BC \rightarrow D \}$, we see that $\{AC\}^+ = ACD$, so $D$ is extraneous in $AC \rightarrow D$.
  - (If you eliminate the entire RHS of a functional dependency, it goes away)
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:

- \( F_c = \{ A \rightarrow B, B \rightarrow C, C \rightarrow A \} \)
- \( F_c = \{ A \rightarrow B, B \rightarrow AC, C \rightarrow B \} \)
- \( F_c = \{ A \rightarrow C, C \rightarrow B, B \rightarrow A \} \)
- \( F_c = \{ A \rightarrow C, B \rightarrow C, C \rightarrow AB \} \)
- \( F_c = \{ A \rightarrow BC, B \rightarrow A, C \rightarrow A \} \)