COMP353 Databases

Schema Refinement:

Minimal Bases

(Canonical Covers)

Minimal Basis (Canonical Cover)

- Recall that the number of iterations to compute the closure of a set of attributes depends on the number of attributes
  - The complexity of some other algorithms which we will study (eg, decomposition algorithms) depend on the number of FD’s
- To ease the situation, can we “minimize” **F**?

Covers/bases

- Note that FD's on a relation may be represented in different but equivalent ways.
- Recall that, given two sets of FD's **F** and **G** on **R,** we say that: "**G** *follows from* **F** (**F** ⊨ **G**)", provided for every instance **r** of **R,** if **r** satisfies **F**, then **r** satisfies **G**. In this case, we may also say: "**F** *implies* **G**", or "**F** covers **G**", *or "***G** *is implied by* **F**"*.*
- If **F** ⊨ **G** and **G** ⊨ **F** both hold, then we say that **G** and **F** are equivalent and denote this by **F ≡ G.** In this case, **we may** also say that **F** and **G** are covers for each other.
- Note that **F ≡ G** iff $\mathbf{F^+} \equiv \mathbf{G^+}$

Canonical Cover (minimal basis)

- Let **F** be a set of FD's. A canonical cover of **F** is a set **G** of FD's that satisfies the following conditions:
  1. **G** is equivalent to **F**, that is, **G** ≡ **F**
  2. Every FD in **G** has a single attribute on the right-hand side.
  3. **G** is minimal, that is, if we obtain a set **H** of FD's from **G** by deleting some FD's in **G** or by reducing the left hand side of some FD's, then **H** won't be equivalent to **F** (that is, **H** ≢ **F)**

Canonical Cover

➔ A canonical cover **G** is minimal in two respects:

1. Every FD in **G** is “required” in order for **G** to be equivalent to **F**
2. Every FD in **G** is as “small” as possible, that is,
   - each attribute on the left hand side **X** is necessary.
   - Recall: the RHS of every FD in **G** is a single attribute

Computing Canonical Cover

Given a set **F** of FD's, how to compute ***a*** canonical cover **G** of **F**?

- **Step 1:** Put the FD's in the **simple** form (i.e., one attribute on the RHS)
  - Initialize **G := F**
  - Replace each FD $\mathbf{X} \longrightarrow \mathbf{A_1A_2 \ldots A_k}$ in **G** with $\mathbf{X \longrightarrow A_1},\ \mathbf{X \longrightarrow A2}, \ldots,\ \mathbf{X \longrightarrow A_k}$
- **Step 2: Minimize** the left hand side **X** of every FD
  - E.g., if AB⟶C is in **G**, check if A or B on the LHS is redundant **,** i.e.,

    $(\mathbf{G} - \{\mathbf{AB} \longrightarrow \mathbf{C}\} \cup \{\mathbf{A} \longrightarrow \mathbf{C}\})^+ \equiv \mathbf{F}^+$ **or**

    $(\mathbf{G} - \{\mathbf{AB} \rightarrow \mathbf{C}\} \cup \{\mathbf{B} \rightarrow \mathbf{C}\})^+ \equiv \mathbf{F}^+$?
- **Step 3: Delete** redundant FD's, if any
  - For each FD $\mathbf{X} \longrightarrow \mathbf{A}$ in **G**, check if $(\mathbf{G} - \{\mathbf{X} \longrightarrow \mathbf{A}\})^+ \equiv \mathbf{F}^+$**?**

Computing Canonical Cover

- **R = { A, B, C, D, E, H}**
- **F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }**

- Step 1 – put FD's in the simple form
  - All present FD's are simple

➔ **G = {A→B, DE→ A, BC → E, AC → E, BCD → A, AED → B}**

Computing Canonical Cover

- **R = { A, B, C, D, E, H }**
- **F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }**
- Step 2 – Check every FD to see if it is left reduced
  - For every FD **X → A** in **G,** check if the closure of a subset of **X** determines **A**. If so, remove the redundant attribute(s) from **X**

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
  - A → B

    ➔ obviously OK (no left redundancy)
  - DE → A
    - $D^+ = D$
    - $E^+ = E$

    ➔ OK (no left redundancy)

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
  - BC → E
    - $B^+ = B$
    - $C^+ = C$

    ➔ OK (no left redundancy)

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
  - AC → E
    - A^+ = AB
    - C^+ = C

    ➔ OK (no left redundancy)

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
  - BCD → A
    - $B^+ = B$
    - $C^+ = C$
    - $D^+ = D$
    - $\{BC\}^+ = BCE$
    - $\{CD\}^+ = CD$
    - $\{BD\}^+ = BD$

➔OK (no left redundancy)

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { A → B, DE → A, BC → E, AC → E, BCD → A, **AED → B** }
  - AED → B
    - $A^+ = AB$
    - E & D are redundant ➔ we can remove them from **AED → B**
- G = { ~~A → B~~, DE → A, BC → E, AC → E, BCD → A, A → B }

➔ G = { DE → A, BC → E, AC → E, BCD → A, A → B }

Computing Canonical Cover

- **R = { A, B, C, D, E, H}**
- **F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }**
- Step 3 – Find and remove redundant FDs
  - For every FD **X → A** in **G**
    - Remove **X → A** from **G**; call the result **G'**
    - Compute $\mathbf{X^+}$ under **G'**
    - If **A** $\in$ $\mathbf{X^+}$, then **X → A** is redundant and hence we remove the FD **X → A** from **G** (that is, we rename **G'** to **G**)

Computing Canonical Cover

- **R = { A, B, C, D, E, H }**
- **F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }**
- **G = { DE → A, BC → E, AC → E, BCD → A, A → B }**
  - Remove **DE → A** from **G**
    - **G' = { BC → E, AC → E, BCD → A, A → B }**
  - Compute **DE**$^+$ under **G'**
    - **{DE}**$^+$ = **DE** (computed under **G'**)
  - Since **A** $\notin$ **DE**, the FD **DE → A** is not redundant
    - **G = { DE → A, BC → E, AC → E, BCD → A, A → B }**

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { DE → A, BC → E, AC → E, BCD → A, A → B }
    - Remove **BC → E** from **G**
        - G' = { DE → A, AC → E, BCD → A, A → B }
    - Compute $\mathbf{BC^+}$ under **G'**
        - $\{BC\}^+ = BC$

          ➔ **BC → E** is not redundant
        - G = { DE → A, BC → E, AC → E, BCD → A, A → B }

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }

- G = { DE → A, BC → E, AC → E, BCD → A, A → B }
  - Remove **AC** → **E** from **G**
    - G' = { DE → A, BC → E, BCD → A, A → B }
  - Compute {**AC**}$^+$ under **G'**
    - {AC}$^+$ = ACBE

  Since E∈ ACBE, AC → E is redundant ➔ remove it from **G**

    - G = { DE → A, BC → E, BCD → A, A → B }

Computing Canonical Cover

- R = { A, B, C, D, E, H }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { DE → A, BC → E, BCD → A, A → B }
  - Remove **BCD → A** from **G**
    - G' = { DE → A, BC → E, A → B }
  - Compute **BCD**$^+$ under **G'**
    - {BCD}$^+$ = BCDEA
  - This FD is redundant ➔ remove it from **G**
    - G = { DE → A, BC → E, A → B }

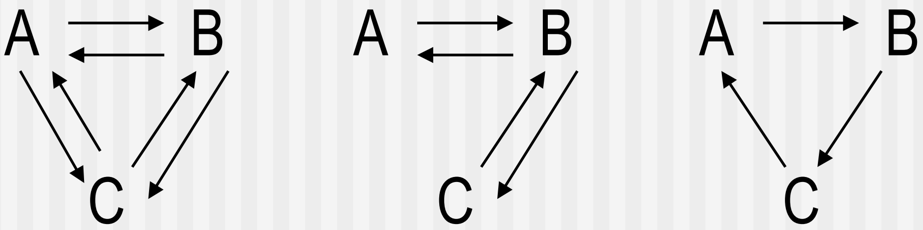
Computing Canonical Cover

- R = { A, B, C, D, E, F }
- F = { A → B, DE → A, BC → E, AC → E, BCD → A, AED → B }
- G = { DE → A, BC → E, A → B }
  - Remove **A → B** from **G**
    - G' = { DE → A, BC → E }
  - Compute **A**$^+$ under **G'**
    - A$^+$ = A
  - This FD is not redundant (Another reason why need **A → B** ?)
    - G = { DE → A, BC → E, A → B }

      ➔ **G** is a minimal cover for **F**

Several Canonical Covers Possible?

- Relation R={A,B,C} with $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, C \rightarrow B, C \rightarrow A\}$
- Several canonical covers exist
  - $G = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}$
  - $G = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$



Can you find more ?

Computing a Canonical Cover

This example shows the order of steps 2 and 3 is important!

R= {**A,B,C,D**} with **F** = { **ABC** → **D**, **AB** → **C**, **D** → **C** }

1. (step 3; step 2): Doing step 3 first, no FD is redundant (Why?)

   In step 2, **ABC** → **D** is left reduced to **AB** → **D**. No more changes.

   We thus obtain **G** = { **AB** → **D**, **AB** → **C**, **D** → **C** } which is equivalent to **F** but is **not** minimal! (The red FD is redundant!).

2. (step 2; step 3): Following our algorithm, in step 2 we get **G** above. In step 3, we remove the redundant FD in **G**.

   This yields { **AB** → **D**, **D** → **C**} which is equivalent to **F** and minimal.

How to Deal with Redundancy?

Relation Schema:

Star (name, address, representingFirm, spokesPerson)

F = { name → address, representingFirm, spokePerson,
representingFirm → spokesPerson }

Relation Instance:

Name	Address	RepresentingFirm	SpokesPerson
Carrie Fisher	123 Maple	Star One	Joe Smith
Harrison Ford	789 Palm dr.	Star One	Joe Smith
Mark Hamill	456 Oak rd.	Movies & Co	Mary Johns

- We can **decompose** this relation into two smaller relations

How to Deal with Redundancy?

Given the relation schema below:

Star (name, address, representingFirm, spokesperson) with

F = {name → address, representingFirm, spokePerson
representingFirm → spokesPerson }

Decompose Star into the following 2 relations:

Star (name, address, representingFirm)
with **F1**={ name → address,representingFirm }

and

Firm (representingFirm, spokesPerson)
with **F2**= { representingFirm → spokesPerson }

How to Deal with Redundancy?

Instance of Star before decomposition:

Name	Address	RepresentingFirm	Spokesperson
Carrie Fisher	123 Maple	Star One	Joe Smith
Harrison Ford	789 Palm dr.	Star One	Joe Smith
Mark Hamill	456 Oak rd.	Movies & Co	Mary Johns

The instance after the decomposition:

Name	Address	RepresentingFirm
Carrie Fisher	123 Maple	Star One
Harrison Ford	789 Palm dr.	Star One
Mark Hamill	456 Oak rd.	Movies & Co

RepresentingFirm	Spokesperson
Star One	Joe Smith
Movies & Co	Mary Johns

Decomposition

- A ***decomposition*** of a relation schema **R** is obtained by splitting **R** into two or more relations, denoted as $\underline{\mathbf{R}} = \{\mathbf{R}_1,\ldots,\mathbf{R}_m\}$. Formally, $\underline{\mathbf{R}}$ is a decomposition of **R** if the following two conditions hold:

  1. No attribute of **R** is lost or introduced (i.e., $\mathbf{R}_1 \cup \ldots \cup \mathbf{R}_m = \mathbf{R}$)
  2. No schema $\mathbf{R}_i$ is a subset or equal to any relation $\mathbf{R}_j$ (for $i \neq j$)

  - When **m = 2**, the decomposition $\underline{\mathbf{R}} = \{ \mathbf{R}_1, \mathbf{R}_2 \}$ is called ***binary***
- Not every decomposition of R is “desirable”. Why?
- Properties of a decomposition?

  (1) Lossless-join – this is a **must**

  (2) Dependency-preserving – this is **desirable**

Explanation follows …

Example

Relation Instance:

A	B	C
1	2	3
4	2	5

Decomposed into:

A	B
1	2
4	2

B	C
2	3
2	5

To “recover” information, we join the relations:

A	B	C
1	2	3
4	2	5
4	2	3
1	2	5

Why do we got new tuples?

Lossless-Join Decomposition

- Suppose **R** is a relation and **F** is a set of FD's over **R**.

  A binary decomposition of **R** into relation schemas **R_1** and **R_2** with attribute sets **X** and **Y** is said to be a **lossless-join decomposition with respect to F**, if for every instance *r* of **R** that satisfies **F**, it holds that $\pi_X(r) \bowtie \pi_Y(r) = r$
- **Thm**: Let **R** be a relation schema and **F** a set of FD's on **R**. A binary decomposition of **R** into **R_1** and **R_2** with attribute sets **X** and **Y** is lossless **if** $X \cap Y \rightarrow X$ or $X \cap Y \rightarrow Y$, i.e., this binary decomposition is lossless if the common attributes of **X** and **Y** form a key of **R_1** or **R_2**

Example: Lossless-join

Relation Instance:

A	B	C
1	2	3
4	2	3

F = { B → C }

Decomposed into:

A	B
1	2
4	2

B	C
2	3

To recover the original relation r, we join the two relations:

A	B	C
1	2	3
4	2	3

No new tuples !

Example: Dependency Preservation

Relation Instance:

A	B	C	D
1	2	5	7
4	3	6	8

F = { B → C, B → D, A → D }

Decomposed into:

A	B
1	2
4	3

B	C	D
2	5	7
3	6	8

Can we enforce A → D?

How ?

Dependency-Preserving Decomposition

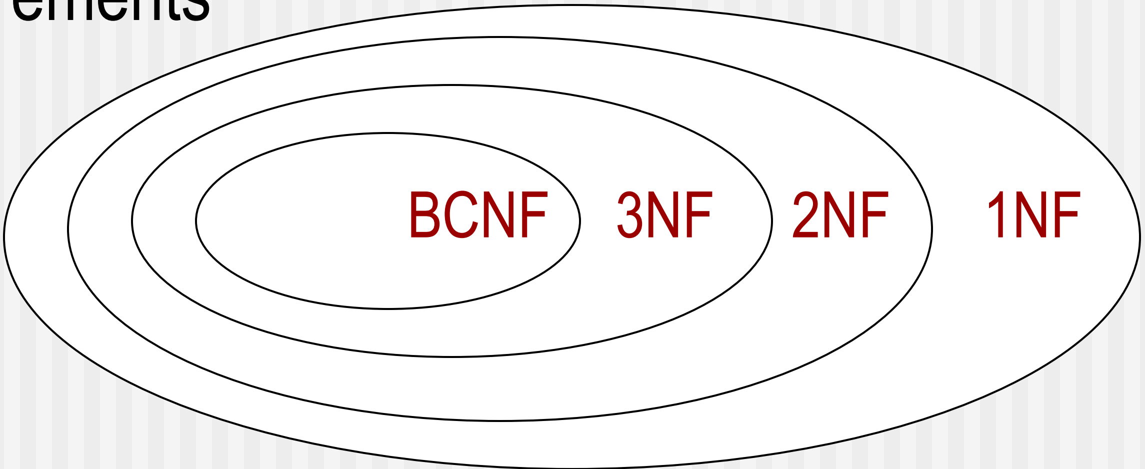
- A dependency-preserving decomposition allows us to enforce every FD (on each insertion of a tuple or when modifying a tuple) by examining just one single relation instance
- Let **R** be a relation schema that is decomposed into two schemas with attribute sets **X** and **Y,** and let **F** be a set of FD's over **R**. The **projection of F on X** (denoted by $\mathbf{F_X}$) is the set of FD's in $\mathbf{F^+}$ that follow from **F** and involve only attributes in **X**
  - Recall that a FD $\mathbf{U} \rightarrow \mathbf{V}$ in $\mathbf{F^+}$ is in $\mathbf{F_X}$ if all the attributes in **U** and **V** are in **X**; In this case, we say this FD is "relevant" to **X**
- The decomposition of < **R, F** > into two schemas with attribute sets **X** and **Y** is **dependency-preserving** if $\mathbf{(F_X \cup F_Y)^+ \equiv F^+}$

Normal Forms

- Given a relation schema **R**, we must be able to determine whether it is "good" or we need to decompose it into smaller relations, and if so, how?
- To address these issues, we need to study **normal forms**
- If a relation schema is in one of these normal forms, we know that it is in some "good" shape in the sense that *certain kinds of problems (related to redundancy) cannot arise*

Normal Forms

- The normal forms based on FD’s are
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
- These normal forms have increasingly restrictive requirements



Third Normal Form (3NF)

Let **R** be a relation schema, **F** a set of FD's on **R**, **X** $\subseteq$ **R**, and **A** $\in$ **R**.

- We say **R** w.r.t. **F** is in 3NF (**third normal form**), if for every FD **X** $\rightarrow$ **A** in **F**, at least one of the following conditions holds:
  - **A** $\in$ **X**, that is, **X** $\rightarrow$ **A** is a trivial FD, or
  - **X** is a superkey, or
  - If **X** is not a key, then **A** is part of some key of **R**
- To determine if a relation <**R, F**> is in 3NF:
  - We check whether the LHS of each nontrivial FD in **F** is a superkey
  - If not, we check whether its RHS is part of any key of **R**

Boyce-Codd Normal Form

Let **R** be a relation schema, **F** a set of FD's on **R**, $\mathbf{X} \subseteq \mathbf{R}$, and $\mathbf{A} \in \mathbf{R}$.

- We say **R** w.r.t. **F** is in **Boyce-Codd normal form**, if for every FD **X** → **A** in **F**, at least one of the following conditions holds:
  - $\mathbf{A} \in \mathbf{X}$, that is, **X** → **A** is a trivial FD, or
  - **X** is a superkey
- To determine whether **R** with a given set of FD's **F** is in BCNF
  - Check whether the LHS **X** of each nontrivial FD in **F** is a superkey
    - How? Simply compute $\mathbf{X}^+$ (w.r.t. **F**) and check if $\mathbf{X}^+ = \mathbf{R}$