Discussion #2

Diwanshu, Elena

June 17, 2020

Logistics

Due this Sunday (6/21):

Homework 2, Checkpoint 1

Project Groups set: Find here

Outline

- Functional Dependency
- Normalization
- Candidate and Super Keys
- Armstrong's Axioms
- Attribute Closure
- Normal Forms
- BCNF, 3NF
- Good Decomposition

Functional Dependency (FD)

- Constraint between two sets of attributes in a relation from a database.
- A → B: Attribute A functionally determines B.
- Eg. campus_id → name

Formal Definition:

- If two tuples agree on the attributes A1, A2, …, An, then they must also agree on the attributes B1, B2, …., Bm.

  A1, A2, …, An → B1, B2, ….. Bm

Candidate Keys and Super Keys

- Set of attributes that functionally determines all attributes of R
- none of its subsets determines all attributes of R
- Super Key is the set that contains candidate keys (or simply, keys).

Q. If |.| denotes cardinality of the set, compare

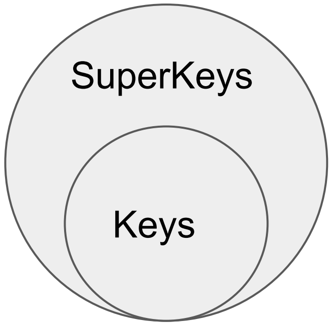
|Super Keys| ? |Candidate Keys|

Candidate Keys and Super Keys

- Set of attributes that functionally determines all attributes of R
- none of its subsets determines all attributes of R
- Super Key is the set that contains candidate keys (or simply, keys).

Q. If |.| denotes cardinality of the set, compare

|Super Keys| >= |Candidate Keys|



A candidate key is a 'minimal' super key.

Inferring all FDs: Armstrong's axioms

S: a set of FDs

S+ = all FDs logically implied by S

Eg. R(A, B, C), S = {A→ C, C→ B}, then S+ = {A → C, C → B, A → B, A → AC, C→ BC, AC → ABC,...}

Inferring all FDs: Armstrong's axioms

Eg. R(A, B, C), S = {A→ C, C→ B}, then S+ = {A → C, C → B, A → B, A → AC, C→ BC, AC → A,...}

Reflexivity rule: X → a subset of X

Augmentation rule: X → Y, then XZ → YZ

Transitivity rule: X → Y and Y → Z then X → Z

Additional Rules: Can be derived from first 3 rules

Union rule

$X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

(X, Y, Z are sets of attributes)

Proof:

Additional Rules: Can be derived from first 3 rules

Union rule

$X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

(X, Y, Z are sets of attributes)

Proof:

1. Augmentation: $X \rightarrow XY$, $XY \rightarrow YZ$
2. Transitivity: $X \rightarrow YZ$

Additional Rules: Can be derived from first 3 rules

Decomposition rule

$X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Pseudo-transitivity rule

$X \rightarrow Y$ and $YZ \rightarrow U$, then $XZ \rightarrow U$

Proof: Exercise.

Inferring S+ using Armstrong’s Axioms

S = S +

- Loop

– For each F in S, apply reflexivity and augmentation rules *(High level Idea: Break & Expand)*

– add the new FDs to S +

– For each pair of FDs in S, apply the transitivity rule *(High level Idea: New FDs)*

– add the new FD to S +

- Until does not change any further

Attribute Closure

Given:

- A set of attributes {A1, A2, …, An}
- A set of FDs S

Closure of {A1, A2, …, An} under FDs S:

- A set of attributes {B1, B2, …, Bm} such that {A1, A2, …, An} → Bi for all i.
- Denoted by {A1, A2, …, An}+

Q. Which attribute should {A1, A2, … An}+ contain at a minimum?

Attribute Closure

Given:

- A set of attributes {A1, A2, …, An}
- A set of FDs S

Closure of {A1, A2, …, An} under FDs S:

- A set of attributes {B1, B2, …, Bm} such that {A1, A2, …, An} → Bi for all i.
- Denoted by {A1, A2, …, An}+

Q. Which attribute should {A1, A2, … An}+ contain at a minimum?

A. {A1, A2, …, An} as {A1, A2, …, An} → Ai is trivial.

Uses of Attribute Closure

- To prove correctness of rules for manipulating FDs.
  - Transitive rule: $A1, A2, \ldots, An \rightarrow B1, B2, \ldots, Bm$ and $B1, B2, \ldots, Bm \rightarrow C1, C2, \ldots, Cp$ then $A1, A2, \ldots, An \rightarrow C1, C2, \ldots, Cp$

Option 1. $\{A1, A2, \ldots, An\} \subseteq \{C1, C2, \ldots, Cp\}^+$

Option 2. $\{C1, C2, \ldots, Cp\} \subseteq \{A1, A2, \ldots, An\}^+$

Option 3. $\{C1, C2, \ldots, Cp\}^+ \subseteq \{A1, A2, \ldots, An\}$

Option 4. $\{A1, A2, \ldots, An\}^+ \subseteq \{C1, C2, \ldots, Cp\}$

Uses of Attribute Closure

- To prove correctness of rules for manipulating FDs.
  - Transitive rule: A1, A2, …, An → B1, B2, …, Bm and B1, B2, …, Bm → C1, C2, …, Cp then A1, A2, …, An → C1, C2, …, Cp

Option 1. {A1, A2, …, An} ⊆ {C1, C2, …, Cp}+

Option 2. {C1, C2, …, Cp} ⊆ {A1, A2, …, An}+

Option 3. {C1, C2, …, Cp}+ ⊆ {A1, A2, …, An}

Option 4. {A1, A2, …, An}+ ⊆ {C1, C2, …, Cp}

Checking X → Y holds: if Y is contained in X+, i.e. **Y ⊆ X+**

Uses of Attribute Closure

- To define Keys
  - In R(A1, A2, …, An), what's the condition for {A1, A2} to be the candidate key?

Option 1. $\{A1, A2\}^+ = \{A1, A2, \ldots, An\}$

Option 2. $\{A1\}^+ = \{A1, A2, \ldots, An\}$

Option 3. $\{A2\}^+ = \{A1, A2, \ldots, An\}$

Option 4. Both 2 and 3.

Uses of Attribute Closure

- To define Keys
  - In R(A1, A2, …, An), what's the condition for {A1, A2} to be the candidate key?

    Option 1. {A1, A2}+ = {A1, A2, …, An}

    Option 2. {A1}+ = {A1, A2, …, An}

    Option 3. {A2}+ = {A1, A2, …, An}

    Option 4. Both 2 and 3.

Uses of Attribute Closure

- To compute closure F+ of FDs F. Eg. R(A,B,C). F = { A → B, B → C}.
- Great example in Slide Set 3: #55, #56 (Monday, June 15).
  - Steps:
    - Construct an empty matrix, with all possible combinations of attributes in col and rows.
    - compute attribute closure for all attribute/ combination of attributes.
    - Fill the matrix. Find FDs.

	A	B	C	AB	AC	BC	ABC
A							
B							
C							
AB							
AC							
BC							
ABC							

Attribute closure
A^+=?
B^+=?
C^+=?
AB^+=?
AC^+=?
BC^+=?
ABC^+=?

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B		√	√			√	
C			√				
AB	√	√	√	√	√	√	√
AC	√	√	√	√	√	√	√
BC		√	√			√	
ABC	√	√	√	√	√	√	√

More Question

- In R(A1, A2, …, A6), functional dependency set F = {$A_i \rightarrow A_{i+2}$} for all i = 1, …, 4. What's the candidate key for R?

  Option 1. {A_1}

  Option 2. {A_1, A_2}

  Option 3. {A_1, A_2, A_3, A_4}

  Option 4. {A_1, A_2, …, A_6}

More Question

- In R(A1, A2, …, A6), functional dependency set F = $\{A_i \rightarrow A_{i+2}\}$ for all i = 1, …, 4. What's the candidate key for R?

  Option 1. $\{A_1\}$

  Option 2. $\{A_1, A_2\}$

  Option 3. $\{A_1, A_2, A_3, A_4\}$

  Option 4. $\{A_1, A_2, \ldots, A_6\}$

- Can you find for general case? i.e. for F = $\{A_i \rightarrow A_{i+m}\}$ for all i = 1, …, n-m.
  - Hint: Think about various permissible values of m.

Normalization

- Process of organizing the attributes of the database to **reduce or eliminate data redundancy** (having the same data but at different places) .

Problems because of data redundancy

- Increases the size of the database
    - Same data repeated at many places
- Inconsistency problems
    - insert, delete and update operations.

Why Normalize?

Sales Staff Table

Employee_ID	SalesPerson	Office Loc.	Phone #	Customer 1	Customer 2	Customer 3
531	Alice	Madison	909-909-2323	American	Delta	United
540	Bob	LA	890-909-3498	Amazon	Microsoft	
558	Chad	Madison	909-909-2323	Ford		

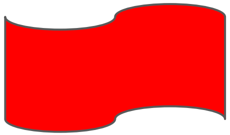
- Good/Bad table?

Why Normalize?

Sales Staff Table

Employee_ID	SalesPerson	Office Loc.	Phone #	Customer 1	Customer 2	Customer 3
531	Alice	Madison	909-909-2323	American	Delta	United
540	Bob	LA	890-909-3498	Amazon	Microsoft	
558	Chad	Madison	909-909-2323	Ford		

- INSERT ANOMALY
  - There are facts we cannot record until we know information for the entire row.
  - Insert new sales office? Need to provide a primary key!

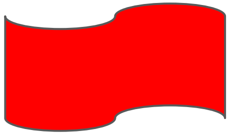


Why Normalize?

Sales Staff Table

Employee_ID	SalesPerson	Office Loc.	Phone #	Customer 1	Customer 2	Customer 3
531	Alice	Madison	909-909-2323	American	Delta	United
540	Bob	LA	890-909-3498	Amazon	Microsoft	
558	Chad	Madison	909-909-2323	Ford		

- UPDATE ANOMALY
  - Same info in several rows.
  - Change office number? Need to update in multiple places!

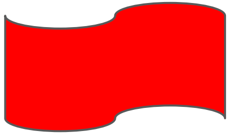


Why Normalize?

Sales Staff Table

Employee_ID	SalesPerson	Office Loc.	Phone #	Customer 1	Customer 2	Customer 3
531	Alice	Madison	909-909-2323	American	Delta	United
540	Bob	LA	890-909-3498	Amazon	Microsoft	
558	Chad	Madison	909-909-2323	Ford		

- DELETE ANOMALY
  - Can lose important information
  - Bob retires? You lose info about LA’s office!

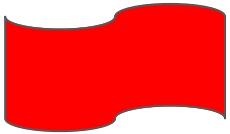


Why Normalize?

Sales Staff Table

<u>Employee_ID</u>	SalesPerson	Office Loc.	Phone #	Customer 1	Customer 2	Customer 3
531	Alice	Madison	909-909-2323	American	Delta	United
540	Bob	LA	890-909-3498	Amazon	Microsoft	
558	Chad	Madison	909-909-2323	Ford		

- <u>REDUNDANCY: FD Exist!</u>
  - Office Loc → Phone #



Normal Forms

A **normal form** represents a “good” schema design.

- **1NF**
  - Every field must contain atomic values (i.e no sets or lists). All relations are in 1NF
- **2NF**
  - All the non-key attributes must depend upon the WHOLE of the candidate key
  - Any relation in 2NF is also in 1NF
- **3NF**
- **BCNF**
- …

more restrictive

Which one we chose?

- We chose the one that is more suitable on the design of our DB and the restrictions that we want to impose to our DB.

What makes a decomposition good? (1)

- minimize redundancy (**the reason why we need the normal forms**)
- avoid information loss (**lossless-join**): I can recover the original data from the decomposed data
- preserve the FDs (**dependency preserving**): no FD is lost
- ensure good query performance

What makes a decomposition good? (2)

Good Example

Person(SSN, name, age, canDrink)

FD:

- *SSN → name, age*
- *age → canDrink*

decomposes into:

- **R1**(SSN, name, age)
  - *SSN → name, age*
- **R2**(age, canDrink)
  - *age → canDrink*

Bad Example

R(A, B, C)

FD:

- A → B
- B, C → A

decomposes into:

- **R1**(A, B)
  - *A → B*
- **R2**(A, C)
  - **no FDs**

R_1

A	B
a_1	b
a_2	b

R_2

A	C
a_1	c
a_2	c

$\bowtie$ *recover*

A	B	C
a_1	b	c
a_2	b	c

The recovered table violates the FD:
B, C → A

Boyce - Codd Normal Form(BCNF)

A relation **R** is in **BCNF** if whenever **X→ A** is a non-trivial FD, then X is a **<u>superkey</u>** in **R**

Trivial FDs:

- Not all FDs are informative:
  - A → A holds for any relation
  - A, B, C → C also holds for any relation
- A FD X→ A is called **<u>trivial</u>** if the attribute A belongs in the attribute set X
  - a trivial FD always holds!

BCNF Example 1

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

Given that:

- **key** = {SSN, phoneNumber}
- **FD**: SSN → name, age

Due to the given FD the above relation is **not** in BCNF! The left part of the FD is not a superkey!

BCNF Example 2

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

Given that:

- **key** = {SSN}
- **FD**: SSN → name, age

Given the key and the FD, the above relation is in BCNF!

How do we know if a relation is in BCNF?

Given a relation R and a set of functional dependency S

1. **Determine a set of candidate keys → Using attribute closure approach**
2. **Check every FD in S to see if the left hand side is a superkey**

Example 1:

Books*(author, gender, booktitle, genre, price)*

FDs:

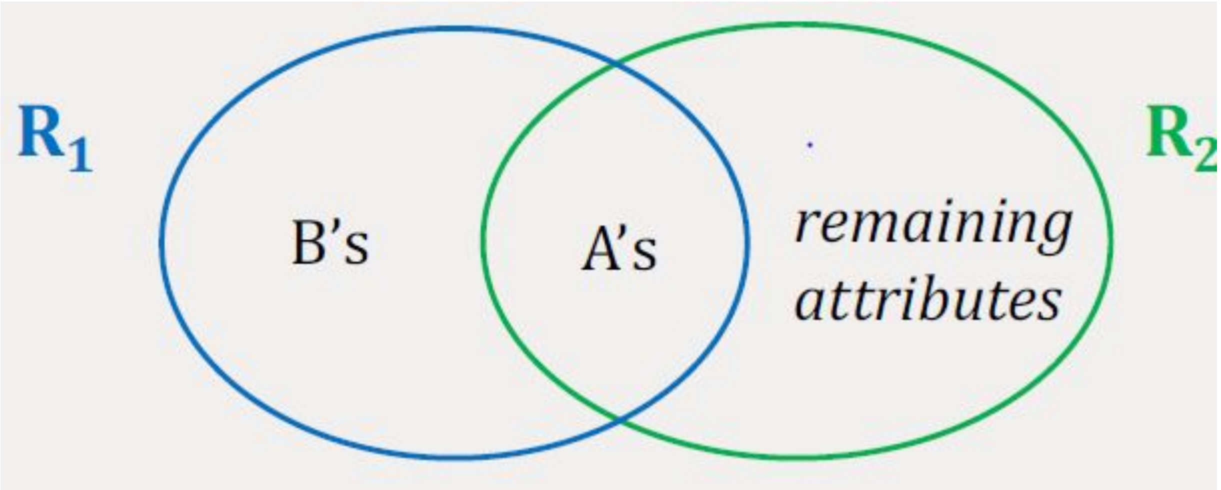
- *author → gender*
- *booktitle → genre, price*

1. **Candidate key:** *(author, booktitle)* and is the only one.
2. **No BCNF**, because the left side of both (no trivial) FDs is not a superkey.

author+ = {author, gender}
gender+ = {gender}
… **(all one attributes)**
(author, gender)+ = {author, gender}
… **(all two attr combinations)**
(author, booktitle)+ = {author, gender, booktitle, genre,price}
… **(all three,... combinations)**

Q: What to do when a relation is NOT BCNF?
A: BCNF Decomposition

1. **Find a FD that violates the BCNF condition**
   A1, A2, …, An → B1, B2, …, Bm
2. **Decompose R to R1 and R2: (see figure)**
3. **Continue until no BCNF violations are left** **(* any 2 attribute relation is in BCNF)**

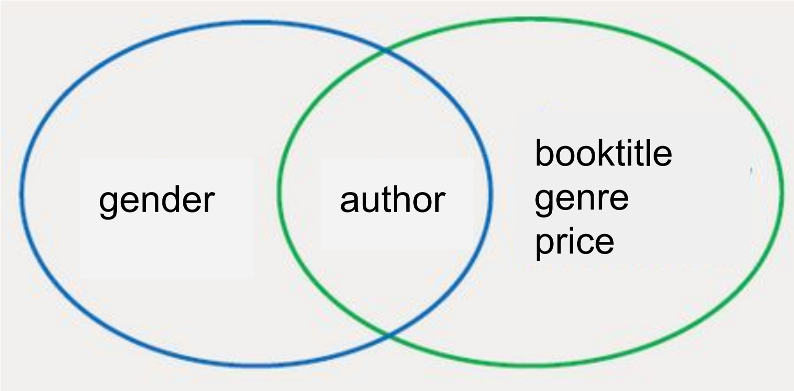


Back in our no BCNF example...

***Books**(author, gender, booktitle, genre, price)*

FDs:

- *author → gender*
- *booktitle → genre, price*



Splitting **Books** using FD *author → gender*

- **Author**(author, gender) with
  FD: *author → gender* in BCNF
- **Books2(**author, booktitle, genre, price) with FD:
  booktitle → genre, price not in BCNF → we need to continue

Back again...

Books*(author, gender, booktitle, genre, price)*

FDs:

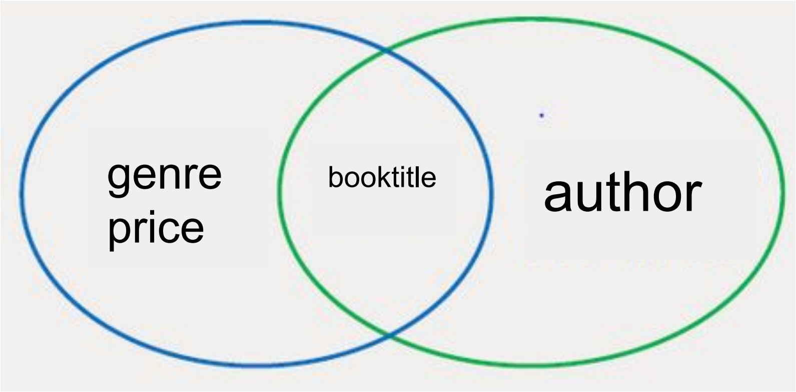
- *author → gender*
- *booktitle → genre, price*

Splitting **Books** using FD *author → gender*

- **Author**(author, gender) <u>with</u>
  FD: *author → gender* in BCNF

Splitting **Books2**(author, booktitle, genre, price):

- **BooksInfo**(booktitle, genre, price) <u>with</u>
  FD: booktitle → genre, price in BCNF
- **BookAuthor**(author, booktitle) in BCNF



Example: Is it a good decomposition?

R(A, B, C)

$S = \{A \rightarrow B, B \rightarrow C\}$

R_1 (A,B) and R_2 (A,C) (not using decomposition heuristic)

– Lossless-join decomposition:

$R1 \cap R2 = \{A\}$ and $A^+=\{A,B\}$, therefore $A \rightarrow AB$

– Dependency preserving

$A \rightarrow B$ is enforced in R_1

but $B \rightarrow C$ is NOT enforced in R_2

Therefore, this decomposition is loss-less join but not dependency preserving

BCNF Decomposition Properties

- removes certain types of redundancy
- is lossless-join
- is **not** always dependency preserving (+ we need to JOIN to find all FDs)

Solution: define a weaker normal form: **3NF**

- allows some redundancy (**that's why is called weaker**) **BUT**
- FDs can be checked on individual relations without computing JOIN
  - JOIN is the one of the most expensive computations in a DB
- There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (3NF)

A relation **R** is in **3NF** if whenever $X \rightarrow A$, one of the following is true:

- $A \in X$ (trivial FD)
- X is a superkey
- A is part of some key of R (prime attribute)

- Informally: everything depends on the key or is in the key
- There is always a lossless-join, dependency-preserving decomposition into 3NF.

3NF vs BCNF

R is in BCNF if whenever $X \rightarrow A$ holds, then X is a superkey.

Slightly stronger than 3NF.

Example:

Given R(A,B,C), a set of FDs
$S = \{AB \rightarrow C, C \rightarrow A\}$

- 3NF but not BCNF

Why? (keys = {AB, BC})

Conclusions

- 3NF is enough, should always aim for this
- BCNF is the most restricted Normal Form
- normalization is not always the solution, we might end up with not a very efficient decomposition that leads to performance loss
    - remember: our goal is good query performance
- current data warehouses argue against normalization
    - joins are expensive or impractical

Q&A

Thanks!