Database Design: Normal Forms

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

October 10, 2018

Announcements

- Project Track 1 – Stage 2

And

- Project Track 2 – Stage 1

are due Today!

- HW 2 is due Oct 15 (23:59)

Overview of Database Design

- Conceptual design: (ER & UML Models are used for this.)
  - What are the entities and relationships we need?
- Logical design:
  - Transform ER design to Relational Schema
- Schema Refinement: (Normalization)
  - Check relational schema for redundancies and related anomalies.
- Physical Database Design and Tuning:
  - Consider typical workloads; (sometimes) modify the database design; select file types and indexes.

Last time!

- ✓ How do we obtain a good design?
- ✓ Functional Dependencies and Keys
- ✓ Rules about Functional Dependencies
  - ✓ Splitting/Combination
  - ✓ Trivial Dependencies
  - ✓ Attribute Closure
  - • FD Closure

Recap: Functional Dependencies

- A form of constraint (hence, part of the schema)
- Finding them is part of the database design
- Used heavily in schema refinement
- Holds for ALL instances!

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots B_m$

Where have we seen this before?

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Recap: Keys are a type of FD

- Key of a relation R is a set of attributes that
  - functionally determines all attributes of R
  - none of its subsets determines all attributes of R
- There could be many keys of a relation
  - Student (UIN, email, dept, age)
  - UIN → UIN, email, dept, age
  - email → UIN, email, dept, age
- Superkey
  - "Superset" of key
  - a set of attributes that contains a key
  - *Any examples for student?*

{UIN, age}

Today!

- Closure of a set of functional dependencies
  - Armstrong's axioms
- Boyce-Codd Normal Form
- Third Normal Form

Closure of a set of FDs

- Given a relation schema R & a set S of FDs
  - Closure of S: S+ = all FDs logically implied by S
  - Allows us to answer all questions of the type
    - is the FD f logically implied by S?
- Example
  - R = {A,B,C,G,H,I}
  - S = A ➔ B; A ➔ C; CG ➔ H; CG ➔ I; B ➔ H
  - would A ➔ H be logically implied?
  - yes (you can prove this, using the definition of FD)
- How to compute S+?

Computing S^+

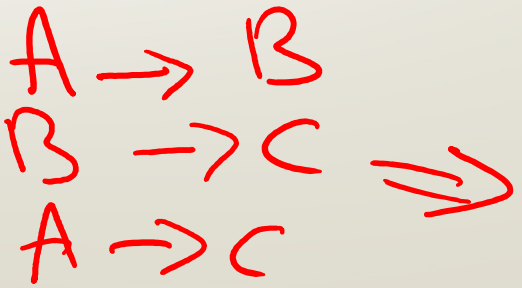
- To check if a specific $A \rightarrow B$ is true, we can compute A^+
  - We already have an algorithm for that

- To compute all $A \rightarrow B$ implied by S, i.e., to compute the closure of S, we can use a particular algorithm.

  - This algorithm depends on the so-called "Armstrong's axioms".
  - These axioms form a *complete* description of FD rules

Armstrong's Axioms

- Reflexivity rule
  - A1A2...An ➔ a subset of A1A2...An
- Augmentation rule
  - A1A2...An ➔ B1B2...Bm, then A1A2...An C1C2...Ck ➔ B1B2...Bm C1C2...Ck
- Transitivity rule
  - A1A2...An ➔ B1B2...Bm and B1B2...Bm ➔ C1C2...Ck, then A1A2...An ➔ C1C2...Ck



Inferring S+ using Armstrong's Axioms

- S+ = S
- Loop until S+ does not change any further
  - For each f in S+, apply reflexivity & augmentation rules
  - Add the new FDs to S+
  - For each pair of FDs in S+, apply the transitivity rule
  - Add the new FD to S+

This procedure could lead to LOTS of rules!!

- You have n attributes, with one FD valid, A → B
- *Exponential in n. Why?*

Two Simple Rules using Armstrong's Axioms

- Armstrong's axioms
  - Reflexivity: A → subset of A
  - Transitivity: A → B, B → C implies A → C
  - Augmentation: A → B implies AC → BC
- Simple Rules
  - Trivial rule – same as reflexivity
  - Combining/splitting rule be expressed as well?
    - X → Y and X → Z, then X → YZ
    - X → YZ then X → Y

Two Simple Rules in Armstrong's Axioms

- Armstrong's axioms
  - Reflexivity: A → subset of A
  - Transitivity: A → B, B → C implies A → C
  - Augmentation: A → B implies AC → BC
- Simple Rules
  - Trivial rule – same as reflexivity
  - Combining rule be expressed as well?
    - To show: X → Y and X →Z, then X → YZ
    - XZ → YZ (aug) and X → XZ (aug)
    - X → YZ (trans)

Two Simple Rules in Armstrong's Axioms

- Armstrong's axioms
  - Reflexivity: A → subset of A
  - Transitivity: A → B, B → C implies A → C
  - Augmentation: A → B implies AC → BC
- Simple Rules
  - Trivial rule – same as reflexivity
  - Splitting rule be expressed as well?
    - To show: X → YZ then X → Y
    - YZ → Y (reflex); then X→Y (trans)

Outline of Rules

- Two examples of rules
  - Splitting/Combination
  - Trivial Dependencies
- Attribute Closure
  - Algorithm
  - Uses
- FD Closure
  - A complete set of rules:
    - Armstrong's axioms
  - Algorithm

But we were talking about schema design

- Armed with the concepts of "functional dependency" and "superkey" (and tools to reason about them), we will now define what a "good schema" is.

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

Whenever there is a nontrivial FD: $A_1A_2\ldots A_n \rightarrow B$,
then $A_1A_2\ldots A_n$ is a super-key for R.

In English (though a bit vague):

Whenever a set of attributes of *R* is determining another attribute, it should determine ***all*** attributes of *R*.

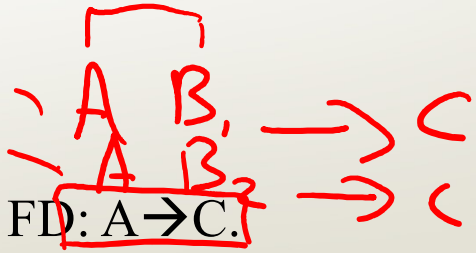
Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

Whenever there is a nontrivial FD: $A_1A_2\ldots A_n \rightarrow B$,
then $A_1A_2\ldots A_n$ is a super-key for R.

Why does this make sense?



Say R(A, B, C) with AB as the key has an FD: A→C.
Then C is being repeated for multiple Bs

Example

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN ➔ Name

Is the left side a superkey?

No

Is it in BCNF?

No.

Decompose it into BCNF

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN → Name

Now is it in BCNF?

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

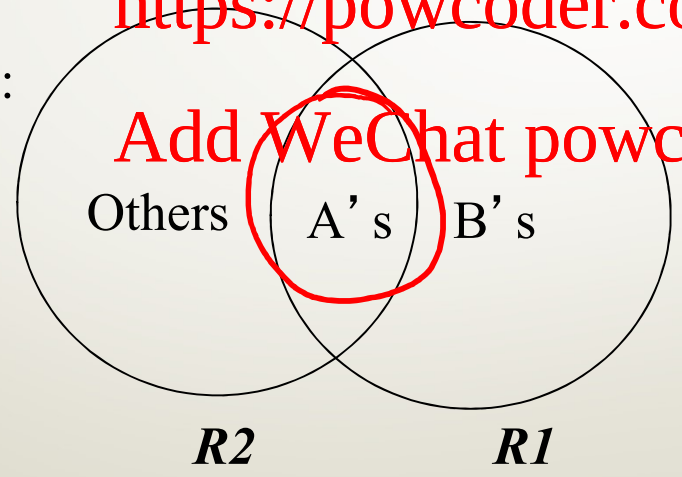
BCNF Decomposition

Find a dependency that violates the BCNF condition:

$A_1, A_2, \ldots A_n \rightarrow B_1, B_2, \ldots B_m$

Heuristic : choose $B_1, B_2, \ldots B_m$ as large as possible

Decompose:



Continue until there are no BCNF violations left.

Example Decomposition

Person:

Name	SSN	Age	EyeColor	PhoneNumber

Functional dependencies:

SSN → Name, Age, Eye Color

BCNF: Person1(SSN, Name, Age, EyeColor),

Person2(SSN, PhoneNumber)

Example

Same example, slightly more complex.

Person (Name, SSN, Age, EyeColor, Phone, Draftworthy)

- FD 1: SSN → Name, Age, EyeColor
- FD 2: Age → Draftworthy

Example

- Person (Name, SSN, Age, EyeColor, Phone, Draftworthy)
- FD 1: SSN → Name, Age, EyeColor
- FD 2: Age → Draftworthy

- FD 1 and 2 imply SSN → Name, Age, EyeColor, Draftworthy
- Split based on this
  - (SSN, Name, Age, EyeColor, Draftworthy)
  - (SSN, Phone Number)
- Split based on Age → Draftworthy
  - (SSN, Name, Age, EyeColor)
  - (Age, Draftworthy)
  - (SSN, Phone Number)

Example

- Movie (title, yr, length, genre, studioName, starName)
- (Title, year, starName) is a key
- FD: Title, year ➔ length, genre, studioName

Example

- Movie (title, yr, length, genre, studioName, starName)
- (Title, year, starName) is a key
- FD: Title, year → length, genre, studioName
  - (Title, Year, Length, Genre, StudioName)
  - (Title, Year, StarName)

Example

- Movie (title, yr, studioName, President, PresAddr)
- FD: Title, yr ➔ studioName
- FD: studioName ➔ President
- FD: President ➔ PresAddr

Example

- Movie (title, yr, studioName, president, presAddr)
- FD: Title, yr ➔ studioName
- FD: studioName ➔ president
- FD: president ➔ presAddr

(title, yr, studioName, president)

(president, presidentaddr)

➔

(title, yr, studioName)

(studioName, president)

(president, presidentaddr)

Two-attribute relations

- Let A and B be the only two attributes of R
- Claim: R is in BCNF. (See Example 3.17.)
- If A ➔ B is true, B ➔ A is not:
- If B ➔ A is true, A ➔ B is not:
- If A ➔ B is true, B ➔ A is true:

Two-attribute relations

- Let A and B be the only two attributes of R

- Claim: R is in BCNF. (See Example 3.17.)

- If A ➔ B is true, B ➔ A is not:
  - A → B does not violate BCNF

- If B ➔ A is true, A ➔ B is not:
  - Symmetric
- If A ➔ B is true, B ➔ A is true:
  - Both are keys, therefore neither violate BCNF

BCNF Decomposition: The Algorithm

Input: relation R, set S of FDs over R

1) Check if R is in BCNF, if not:

   a) pick a violation FD f: A → B

   b) compute A+

   c) create R1 = A+, R2 = A union (R – A+)

   d) compute all FDs over R1, using R and S. Repeat similarly for R2. (See Algorithm 3.12)

   e) Repeat Step 1 for R1 and R2

2) Stop when all relations are BCNF, or are two-attributes

(Remember, two attribute relations are always in BCNF)

Q: Is BCNF Decomposition unique?

- R(SSN, netid, phone).
- FD1: SSN -> netid
- FD2: netid -> SSN
- Each of these two FDs violates BCNF.

Can you tell me two different BCNF decomp for R?

- Pick FD1 and decompose, you get:
  - (SSN, netid); (SSN, phone).
- Pick FD2 and you get
  - (netid, SSN); (netid, phone).

Properties of BCNF

- Property 1. BCNF removes certain types of redundancies
  - those caused by adding many-many or one-many relationships

- For examples of redundancy that it cannot remove, see "multi-valued redundancy"
  - But this is not in curriculum

TB (3.6)

Properties of BCNF

- BCNF Decomposition avoids information loss
  - You can construct the original relation instance from the decomposed relations' instances.

  - *How would get R(A, B, C) from R(A, B), R(B, C)?*
  - *Ans: Natural join*

An easy decomposition?

- We saw that two-attribute relations are in BCNF.
- Why don't we break any R(A,B,C,D,E) into R1(A,B); R2(B,C); R3(C,D); R4(D,E)? Why bother with finding BCNF violations etc.?
- *Turns out, this leads to information loss …*

Example of the "easy decomposition"

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

R1

A	B
1	2
4	2

R2

B	C
2	3
2	6

Example of the "easy decomposition"

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

Nat.Join

A	B
1	2
4	2

B	C
2	3
2	6

A	B	C
1	2	3
4	2	6
1	2	6
4	2	3

We get back some "bogus tuples" !

BCNF Decomposition is Lossless

Example R(A, B, C), A -> C

BCNF: R1(A,B), R2(A,C)

Some tuple (a,b,c) in R

decomposes into (a,b) in R1

and (a,c) in R2

Recover tuples in R: (a,b,c),

BCNF Decomposition is Lossless

Example R(A, B, C), A -> C

BCNF: R1(A,B), R2(A,C)

Some tuple (a,b,c) in R (a,b',c') also in R

decomposes into (a,b) in R1 (a,b') also in R1

and (a,c) in R2 (a,c') also in R2

Recover tuples in R: (a,b,c),

BCNF Decomposition is Lossless

Example R(A, B, C), A -> C

BCNF: R1(A,B), R2(A,C)

Some tuple (a,b,c) in R (a,b',c') also in R

decomposes into (a,b) in R1 (a,b') also in R1

and (a,c) in R2 (a,c') also in R2

Recover tuples in R: (a,b,c), (a,b,c'), (a,b',c), (a,b',c') also in R

Is any of these a "bogus tuple" (not present in R)?

BCNF Decomposition is Lossless

Example R(A, B, C), A -> C

BCNF: R1(A,B), R2(A,C)

Some tuple (a,b,c) in R (a,b',c') also in R

decomposes into (a,b) in R1 (a,b') also in R1

and (a,c) in R2 (a,c') also in R2

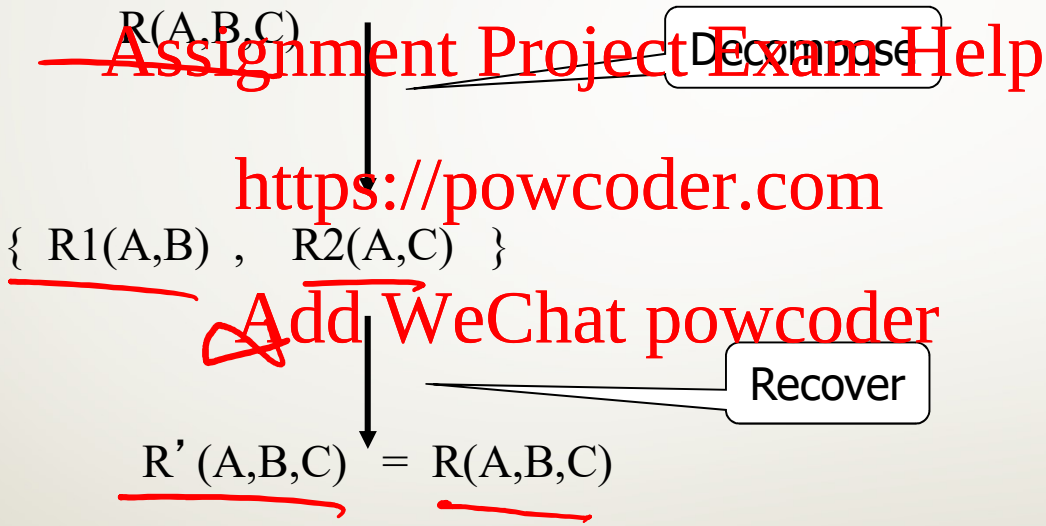
Recover tuples in R: (a,b,c), (a,b,c'), (a,b',c), (a,b',c') also in R

Is any of these a "bogus tuple" (not present in R)?

No! Also see text 3.4.1 for proof. (Essentially an extension of this arg)

Lossless Decompositions

A decomposition is *lossless* if we can recover:



R' is in general larger than R. Why?

Must ensure R' = R

Desirable Properties of Schema Refinement

- ✔ 1) minimize redundancy
- ✔ 2) avoid info loss
- 3) preserve dependency
- 4) ensure good query performance

However,

- BCNF is not always dependency preserving
- In fact, some times we cannot find a BCNF decomposition that is dependency preserving
- Can handle this situation using 3NF
- But what is "dependency preserving"?

Normal Forms

First Normal Form = all attributes are atomic

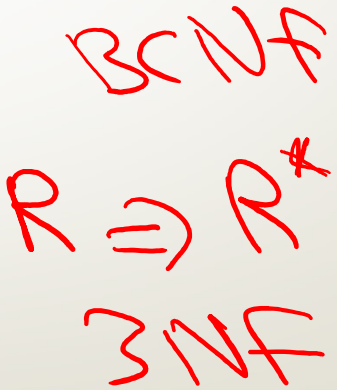
Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...



3NF: A Problem with BCNF

Phone	Address	Name

FD's: Phone -> Address; Address, Name -> Phone

So, there is a BCNF violation (Phone ➔ Address), and we decompose.

Phone	Address

Phone -> Address

Phone	Name

No FDs

So where's the problem?

Phone	Address
1234	10 Downing
5678	10 Downing

Phone	Name
1234	John
5678	John

FD's: Phone-> Address; Address, Name -> Phone

No problem so far. All *local* FD's are satisfied.

Let's put all the data into a single table:

Phone	Address	Name
1234	10 Downing	John
5678	10 Downing	John

Violates the dependency: Address, Name ➔ Phone