CS 4604: Introduction to Database Management Systems

Functional Dependencies

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

VIRGINIA TECH™

Today’s Topics

- Functional dependencies (FD)
  - Definition
  - Armstrong’s “axioms”
  - FD closure and cover
  - Attribute closure
  - (Super)key and candidate key

Steps in Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - *high level description (often done w/ER model)*
  - ORM encourages you to program here

  Completed
- Logical Design
  - translate ER into DBMS data model
  - ORMs often require you to help here too

  Completed
- **Schema Refinement**
  - **consistency, normalization**

  Today
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

Bad Relation Converted from E/R Diagram

- Hard to use (CRUD)
- Mental effort (Treat others are mind readers)
- Arbitrarily (No rules followed)
- **Redundancy** (Space, Inconsistencies, etc.)

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

- One person may have multiple phones, but lives in only one city
- Primary key is what?
- What is the problem with this schema?

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Oakland”?
- Deletion anomalies = what if Joe deletes his phone number?

Relational Schema Design

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

Name	SSN	City
Fred	123-45-6789	Berkeley
Joe	987-65-4321	San Jose

SSN	PhoneNumber
123-45-6789	510-555-1234
123-45-6789	510-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to "Oakland"
- Easy to delete all Joe's phone numbers

Relational Schema Design (or Logical Design)

- How do we do this **systematically**?
  - – Start with some relational schema
  - – Find out its ***functional dependencies*** (FDs)
  - – Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

- $X \rightarrow Y$: 'X' functionally **determines** 'Y'
- Informally: 'if you know 'X', there is only one 'Y' to match'
- If t is a tuple in a relation R and A is an attribute of R, then t[A] is the value of attribute A in tuple t

EmpID	Name	Phone	Position
E0045	**Smith**	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Functional Dependencies (FDs)

Formally: $X \rightarrow Y \Rightarrow (t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y])$

if two tuples agree on the 'X' attribute, they ***must*** agree on the 'Y' attribute, too (eg., if ids are the same, so should be names)

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Functional Dependencies (FDs)

X and Y can be **sets** of attributes

A FD on a relation R is a statement:

- If two tuples in R agree on attributes A1, A2, …, An then they must also agree on the attribute B1, B2, …., Bm
- Notation: A1,A2,… An → B1, B2, …, Bm

Functional Dependencies (FDs)

- A FD is a constraint on a single relational schema
  - It must hold on every instance of the relation
  - You can not deduce an FD from a relation **instance**!
  - But you can deduce if an FD does NOT hold using an **instance**

FD Example

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

~~1234 → Clerk~~

~~1234 → Lawyer~~

FD Example - 2

X	Y
1	6
1	7
2	8

$X \rightarrow Y$

X	Y
1	3
1	3
1	3
2	3
3	4

$X \rightarrow Y$

X	Y
1	3
1	3
1	3
2	3
3	4
1	4

$X \rightarrow Y$

FD Summary

- FD **holds** or **does not hold** on an instance
- If we can be sure that every instance of R will be one in which a given FD is true, then we say that R **satisfies** the FD
- If we say that R **satisfies** an FD, we are stating a **constraint** on R

Why We Need FDs?

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

~~Anomalies:~~

- ~~Redundancy = repeat data~~
- ~~Update anomalies = what if Fred moves to "Oakland"?~~
- ~~Deletion anomalies = what if Joe deletes his phone number?~~

An Interesting Observation

- Workers(<u>ssn</u>, name, lot, did, since)
- If these FDs are true:
  - ssn → did
  - did → lot
- Then this FD also holds:
  - ssn → lot

An Interesting Observation - 2

- If all these FDs are true:
  - name → color
  - category → department
  - color, category → price
- Then this FD also holds:
  - name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies!
There could be more FDs implied by the ones we have.

Finding New FDs: Armstrong's Axioms (AA)

- Suppose X, Y, Z are sets of attributes, then:
  - *Reflexivity*: If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- **Sound** and **complete** inference rules for FDs!
- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudo-transitivity: If $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$

Armstrong's Axioms

Prove 'Union' from three axioms:

$$\left.\begin{array}{ll} X \rightarrow Y & (1) \\ X \rightarrow Z & (2) \end{array}\right\}$$

$$(1) + augm.w/Z \Rightarrow XZ \rightarrow YZ \qquad (3)$$

$$(2) + augm.w/X \Rightarrow XX \rightarrow XZ \qquad (4)$$

$$but \quad XX \;\; is \;\; X \quad thus$$

$$(3) \; + \; (4) \;\; and \;\; transitivity \;\; \Rightarrow X \rightarrow YZ$$

Armstrong's Axioms

Prove Decomposition:

$$X \rightarrow YZ \Rightarrow \left. \begin{matrix} X \rightarrow Y \\ X \rightarrow Z \end{matrix} \right\}$$

$YZ \rightarrow Y$ (Reflexivity)

X → Y, So does X → Z

Armstrong's Axioms

Prove Pseudo-transitivity:

$$\left.\begin{array}{l} X \to Y \\ YW \to Z \end{array}\right\} \Rightarrow XW \to Z$$

XW → YW *Augmentation*

XW → Z Transitivity

Example

- Relation R: { A, B, C }
- F = { A → B and B → C }
- FDs
  - A → C
  - AC → BC
  - AB → AC
  - AB → CB
  - AC → B
  - …

Closure of a set of FDs

- Given a set F of FDs, the set of all FDs is called the **closure of F**, denoted as $\mathbf{F^+}$
- Use Armstrong's Axioms to find F^+
- Trivial FD: using reflexivity to generate all trivial dependencies
- Non-trivial FD:
  - Using transitivity and augmentation

Examples of Computing Closures of FDs

- Let us include only completely non-trivial FDs in these examples, with a single attribute on the right
- $F = \{A \rightarrow B, B \rightarrow C\}$
  - $\{F\}+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, AC \rightarrow B, AB \rightarrow C\}$
- $F = \{AB \rightarrow C, BC \rightarrow A, AC \rightarrow B\}$
  - $\{F\}+ = \{AB \rightarrow C, BC \rightarrow A, AC \rightarrow B\}$
- $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$
  - $\{F\}+ = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, A \rightarrow C, A \rightarrow D, B \rightarrow D, \ldots\}$

FDs - ‘canonical cover’ Fc

Given a set F of FD (on a schema)

Fc is a **minimal set** of equivalent FDs. Eg.,

takes(ssn, c-id, grade, name, address)

$$\left.\begin{array}{l}\text{ssn, c-id -> grade} \\ \text{ssn-> name, address} \\ \text{ssn,name-> name, address} \\ \text{ssn, c-id-> grade, name}\end{array}\right\} F$$

FDs - 'canonical cover' Fc

Fc

ssn, c-id -> grade

ssn-> name, address

ssn,name-> name, address

ssn, c-id-> grade, name

F

FDs - ‘canonical cover’ Fc

why do we need it?

– easier to compute candidate keys

define it properly

compute it efficiently

FDs - ‘minimal cover’ Fc

define it properly - three properties

- 1) the RHS of every FD is a single attribute
- 2) the closure of Fc is identical to the closure of F (ie., Fc and F are equivalent)
- 3) Fc is minimal (ie., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated

FDs - ‘minimal cover’ Fc

#3: we need to eliminate ‘extraneous’ attributes. An attribute is ‘extraneous if

- the closure is the same, before and after its elimination
- or if F-before implies F-after and vice-versa

FDs - ‘minimal cover’ Fc

ssn, c-id -> grade

ssn-> name, address

ssn,~~name~~-> name, address

ssn, c-id-> grade, ~~name~~

F

FDs - ‘minimal cover’ Fc

Algorithm:

examine each FD; drop extraneous LHS or RHS attributes; or redundant FDs

make sure that FDs have a single attribute in their RHS

repeat until no change

FDs - ‘minimal cover’ Fc

Trace algo for

AB->C (1)

A->BC (2)

B->C (3)

A->B (4)

FDs - ‘minimal cover’ Fc

Trace algo for

AB->C (1)

A->BC (2)

B->C (3)

A->B (4)

split (2):

AB->C (1)
A->B (2’)
A->C (2’’)
B->C (3)
A->B (4)

FDs - ‘minimal cover’ Fc

AB->C (1)
~~A->B   (2’)~~
A->C   (2’’)
B->C   (3)
A->B   (4)

AB->C (1)

A->C   (2’’)
B->C   (3)
A->B   (4)

FDs - ‘minimal cover’ Fc

AB->C (1)

A->C (2’’)
B->C (3)
A->B (4)

(2’’): redundant (implied by (4), (3) and transitivity

AB->C (1)

B->C (3)
A->B (4)

FDs - ‘minimal cover’ Fc

AB->C (1)

B->C (3)
A->B (4)

in (1), ‘A’ is extraneous:
(1),(3),(4) imply
(1’),(3),(4), and vice versa

B->C (1’)

B->C (3)
A->B (4)

FDs - ‘minimal cover’ Fc

~~B->C (1’)~~

B->C (3)
A->B (4)

$\boxed{\begin{array}{ll} B\text{->}C & (3) \\ A\text{->}B & (4) \end{array}}$

- nothing is extraneous
- all RHS are single attributes
- final and original set of FDs are equivalent (same closure)

FDs - ‘minimal cover’ Fc

BEFORE

AB->C (1)
A->BC (2)
B->C (3)
A->B (4)

AFTER

B->C (3)
A->B (4)

Attributes Closure

- If we just want to check whether a given dependency $X \rightarrow Y$ is in the closure of a set F of FDs
  - We can just compute the attribute closure X^+ without computing F^+
- Compute attribute closure X^+ with respect to F
  - X^+ is the set of attributes A such that $X \rightarrow A$ is in F^+

Closure of Attributes

Given (INPUT) :

- Attributes {A1, A2, .. An}
- Set of FDs S

Find (OUTPUT) :

- $X = \{A1, A2, \dots, An\}^{+}$

Closure Algorithm

$X=\{A_1, \ldots, A_n\}$.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$
{ name, category, color, department, price }

Hence: $\boxed{\text{name, category} \rightarrow \text{color, department, price}}$

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$,

Example:

1. name → color
2. category → department
3. color, category → price

Closures:

$name^+ = \{name, color\}$

$\{name, category\}^+ = \{name, category, color, department, price\}$

$color^+ = \{color\}$

Example

- Relation R: { A, B, C, D, E }
- F = { B $\rightarrow$ CD, D $\rightarrow$ E, B $\rightarrow$ A, E $\rightarrow$ C, AD $\rightarrow$ B }
- Is B $\rightarrow$ E in F^+?
  - evaluate the closure of B
  - B $\rightarrow$ CD, D $\rightarrow$ E
  - B^+ = {B, C, D, E, ….}
  - Thus B $\rightarrow$ E

Definition of Keys

- FDs allow us to formally define keys
- A set of attributes {A1, A2, ..., An} is a key for relation R if:
  - **Uniqueness:** {A1, A2, ..., An} functionally determine all the other attributes of R
  - **Minimality:** no proper set of {A1, A2, ..., An} functionally determines all other attributes of R

Definition of Keys

- A superkey is a set of attributes that has the uniqueness property but is not necessarily minimal
  - A1, ..., An → B
- A **candidate key**(or sometimes just key) is a **minimal** superkey
- If a relation has multiple keys, specify one to be primary key
- If a key has only one attribute A, say A rather than {A}

Computing (Super) Keys

- For all sets X, compute X+
- If X^+ = [all attributes], then X is a superkey
- Try reducing to the minimal X's to get the candidate key

Example

- Product(name, price, category, color)
- FDs
  - name, category → price
  - category → color
- Candidate key:
  - (name, category)+ = { name, category, price, color }
  - Hence (name, category) is a candidate key

Closures of Attributes vs Closure of FDs

Both algorithms take as input a relation R and a set of FDs F

Closure of FDs:

- Computes {F}+, the **set of all FDs** that follow from F
- Output is a set of FDs
- Output may contain an exponential number of FDs

Closure of attributes:

- In addition, takes a set {A1, A2..., An} of attributes as input
- Computes {A1, A2, ..., An}+, the **set of all attributes** B, such that A1 A2 … An → B follows from F
- Output is set of all attributes
- Output may contain at most the number of attributes in R

Example

- Relation R: { A, B, C, D, E }
- F = { B → CD, D → E, B → A, E → C, AD → B }
- Is D a superkey?
  - evaluate the closure of D
  - B → CD, D → E
  - D^+ = {D, E, C}
- Is AD a superkey?
  - evaluate the closure of AD
  - AD → B,
  - AD^+ = {A, D, E, C, B}
- Is AD a candidate key?
- Is ADE a candidate key?

Example 2

- Relation R: { C, S, J, D, P, Q, V }
- F = { JP → C, SD → P, C → CSJDPQV }
- Is SDJ is a key?
  - JP → C, C → CSJDPQV, so JP is a key
  - SD → P, so SDJ → JP
  - So SDJ → CSJDPQV
- Is SD → CSDPQV ?

Reading and Next Class

- Functional Dependencies: Ch 19.1-19.3
- Next: BCNF, 3NF and Normalization: Ch 19.4-19.9