CS460: Intro to Database Systems

Class 6: *Functional Dependencies*

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/

Review: Database Design

Requirements Analysis

user needs; what must database do?

Conceptual Design

high level description (often done w/ ER model)

Logical Design

translate ER into DBMS data model

Schema Refinement

consistency, normalization

Physical Design

indexes, disk layout

Review: Database Design

Requirements Analysis

user needs; what must database do?

Conceptual Design

high level description (often done w/ ER model)

Logical Design

translate ER into DBMS data model

Schema Refinement

consistency, normalization

Physical Design

indexes, disk layout

Why schema refinement

what is a bad schema?

a schema with redundancy!

why?

redundant storage & insert/update/delete anomalies

how to fix it?

normalize the schema by decomposing

normal forms: BCNF, 3NF, … [next time]

Motivating Example

SSN	Name	Salary	Telephone
987-00-8761	John	65K	857-555-1234
987-00-8761	John	65K	857-555-8800
123-00-9876	Anna	80K	617-555-9876
787-00-4321	Kurt	25K	617-555-3761

primary key? 

(SSN,Telephone)

problems of the schema? 

Motivating Example

SSN	Name	Salary	Telephone
987-00-8761	John	65K	857-555-1234
987-00-8761	John	65K	857-555-8800
123-00-9876	Anna	80K	617-555-9876
787-00-4321	Kurt	25K	617-555-3761

Problems

Storage

Update

Insert

Delete

Motivating Example

SSN	Name	Salary	Telephone
987-00-8761	John	65K	857-555-1234
987-00-8761	John	65K	857-555-8800
123-00-9876	Anna	80K	617-555-9876
787-00-4321	Kurt	25K	617-555-3761

Problems

Storage: store *Salary* multiple times

Update: change John’s salary?

Insert: how to store someone with *no phone*?

Delete: how to delete Kurt’s phone?

Solution: Decomposition

SSN	Name	Salary	Telephone
987-00-8761	John	65K	857-555-1234
987-00-8761	John	65K	857-555-8800
123-00-9876	Anna	80K	617-555-9876
787-00-4321	Kurt	25K	617-555-3761

SSN	Name	Salary
987-00-8761	John	65K
123-00-9876	Anna	80K
787-00-4321	Kurt	25K

SSN	Telephone
987-00-8761	857-555-1234
987-00-8761	857-555-8800
123-00-9876	617-555-9876
787-00-4321	617-555-3761

can decomposition cause problems?

how to find good decompositions?

FUNCTIONAL DEPENDENCIES

Functional Dependencies

Definition

Functional Dependencies (FDs) : form of constraint
"generalized keys"

let *X,Y* nonempty sets of attributes of relation R
let t_1, t_2 tuples : if $t_1.X = t_2.X$, then $t_1.Y = t_2.Y$

"X → Y" : "X (functionally) determines Y"

an FD comes from the application (not the data)
an FD cannot be inferred (only validated)

Functional Dependencies

SSN	Name	Salary	Telephone
987-00-8761	John	65K	857-555-1234
987-00-8761	John	65K	857-555-8800
123-00-9876	Anna	80K	617-555-9876
787-00-4321	Kurt	25K	617-555-3761

which attribute determines which?

~~*SSN → Telephone*~~

they both *might* be true:

SSN → Name, Salary

SSN, Salary → Name

actually, the second can be inferred if the first is true! (more on that later)

FD: Example 3

studentID	classID	Semester	Instructor
1234	15	2	Mark
0043	15	1	Evimaria
4322	15	2	Mark
9876	175	4	Dora
1211	177	4	Manos
0043	154	2	Abraham

which attribute determines which?

classID, Semester → Insructor

~~*studentID → Semester*~~

studentID, classID → Semester

Reasoning about FDs

an FD holds for *all* allowable relations (*legal*)
identified based on semantics of application

given an instance r of R and an FD f:
(1) we can check whether the instance r **violates** FD f
(2) we **cannot** determine if f **holds**

"K $\rightarrow$ all attributes of R" then K is a *superkey* for R
(does not require K to be *minimal*)
remember: in order to be a **candidate key** minimality is required

FDs are a generalization of keys

Reasoning about FDs (Splitting)

assume A, B $\rightarrow$ C, D

C, D are *independently determined* by A,B

so, we can split: A, B $\rightarrow$ C and A, B $\rightarrow$ D

it does not work vice versa

we *cannot* infer: A $\rightarrow$ C, D or B $\rightarrow$ C, D

Trivial FDs

for every relation

$A \rightarrow A$

$A, B, C \rightarrow A$

these are not informative!

in general an FD $X \rightarrow A$ is called ***trivial*** if $A \subseteq X$

it always holds!

Identifying FDs

FD comes from the application (domain)

property of app semantics (not of instance)

cannot infer from an instance

given a set of tuples (instance r), we can:

(1) confirm that an FD **might be** valid

(2) infer that an FD is **definitely invalid**

but we cannot prove that an FD is valid

FD: Example 3

name	category	color	price	department
iPhone	smartphone	black	600	phones
Lenovo Yoga	laptop	grey	800	computers
unifi	networking	white	150	computers
unifi	cables	white	10	stationary
OnePlus	smartphone	silver	450	phones

~~name → *department*~~

name, category → *department*

maybe! we do not know!

Why use FDs?

the capture (and generalize) key constraints

offer more integrity constraints

help us detect redundancies
tell us how to normalize

it is the principled way to solve the redundancy problem

More on: Reasoning for FD

when a set of FD holds over a relation

more FD can be inferred

Armstrong’s Axioms

Axiom 1: Reflexivity

for every subset $X \subseteq \{A_1, \ldots, A_n\}$

$$A_1, \ldots, A_n \rightarrow X$$

Examples

$A, B \rightarrow B$

$A, B, C \rightarrow B, C$

$A, B, C \rightarrow A, B, C$

Axiom 2: Augmentation

for any attribute sets X, Y, Z

if X → Y, then X, Z → Y, Z

Examples

A → B then A, C → B, C

A, B → C then A, B, C → C

(here X=A,B and Y=Z=C)

Axiom 3: Transitivity

for any attribute sets X, Y, Z

if X → Y and Y → Z then X → Z

Examples

A → B and B → C then A → C

A → B, C and B, C → D then A → D

Union and Decomposition

rules that follow from AA

Union

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

Decomposition

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

Applying AA

Product

name	category	color	price	department

we know:

(1) name $\rightarrow$ color
(2) category $\rightarrow$ department
(3) color, category $\rightarrow$ price

can we infer: name, category $\rightarrow$ price

(i) augmentation to (1):
(4) name, category $\rightarrow$ color, category
(ii) transitivity to (4), (3)
name, category $\rightarrow$ price

Applying AA

Product

name	category	color	price	department

we know:

(1) name → color
(2) category → department
(3) color, category → price

can we infer: name, category → color

(i) by reflexivity:
(5) name, category → name
(ii) transitivity to (5), (1)
name, category → color

FD Closure

how can we find all FD?

FD Closure

if F is a set of FD, the closure F^+ is the set of all FDs logically implied by F

Using Armstrong Axioms we can find F^+

sound: any generated FD belongs to F^+

complete: repeated application of AA generates F^+

Attribute Closure

X an attribute set, the closure X^+ is
the set of all attributes B : $X \rightarrow B$

in other words :
attribute closure of X is the set of all attributes that
“are (functionally) determined by X”

Applying AA

Product

name	category	color	price	department

we know:

(1) name $\rightarrow$ color

(2) category $\rightarrow$ department

(3) color, category $\rightarrow$ price

Attribute closure:

(i) Closure of name

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

(i) Closure of name, category

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

Calculating Attribute Closure

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

closure = X

UNTIL *closure* does not change **REPEAT**:

    IF $B_1, \ldots, B_m \rightarrow C$ **AND**

        $B_1, \ldots, B_m$ are all in *closure*

    THEN add C to *closure*

Calculating Attribute Closure

Example: R(A,B,C,D,E,F)

$A, B \rightarrow C$

$A, D \rightarrow E$

$B \rightarrow D$

$A, F \rightarrow B$

$\{A,B\}^+$

$\{A,F\}^+$

Calculating Attribute Closure

Example: R(A,B,C,D,E,F)

~~$A, B \rightarrow C$~~

~~$A, D \rightarrow E$~~

~~$B \rightarrow D$~~

$A, F \rightarrow B$

{A,B}

{A,B,C}

{A,B,C,D}

{A,B,C,D,E}

$\{A,B\}^+$

$\{A,F\}^+$

Calculating Attribute Closure

Example: R(A,B,C,D,E,F)

~~A, B → C~~

~~A, D → E~~

~~B → D~~

~~A, F → B~~

$\{A,B\}^+$

$\{A,F\}^+$

{A,B}

{A,B,C}

{A,B,C,D}

{A,B,C,D,E}

{A,F}

{A,F,B}

{A,F,B,C}

{A,F,B,C,D}

{A,F,B,C,D,E}

Calculating Attribute Closure

Example: R(A,B,C,D,E,F)

$A, B \rightarrow C$

$A, D \rightarrow E$

$B \rightarrow D$

$A, F \rightarrow B$

$\{A,B\}^+ = \{A,B,C,D,E\}$

$\{A,F\}^+ = \{A,F,B,C,D,E\}$

Why calculate attribute closure?

for "does $X \rightarrow Y$ hold" questions check if $Y \subseteq X^+$

to compute the closure F^+ of FDs

(i) for each subset of attributes X, compute X^+

(ii) for each subset of attributes $Y \subseteq X^+$, output the FD $X \rightarrow Y$

why do we need the FD closure?

to decide on decomposition (next time)

FD and Keys

in terms of relational model

superkey: a set of attributes such that:

no two distinct tuples can have same values in all key fields

in terms of FD

superkey: a set of attributes $A_1, A_2, \ldots, A_n$ such that
for *any* attribute B: $A_1, A_2, \ldots, A_n \rightarrow B$

key (or candidate key): requires minimality

what if we have multiple candidate keys?

- we specify one to be the **primary key**

Computing (Super)Keys

(1) compute X^+ for all sets of attributes X

(2) if X^+=all attributes, then X is a *superkey*

why?

- because then ”X determines `all attributes`”

(3) if, also, *no subset of X is superkey*

then X is <u>also a key</u>

Example

Product

name	category	color	price

we know:

(1) name $\rightarrow$ color

(2) color, category $\rightarrow$ price

Superkeys:

{name, category}, {name, category, price},
{name, category, color}, {name, category, price, color}

Keys:

{name, category}

Can we have more than 1 key?

what about the relation ***R*** *(A,B,C)* with:

A, B $\rightarrow$ C

A, C $\rightarrow$ B

which are the keys?

{A, B} and {A, C} are both minimal

Should we use all FDs?

given a set of FDs *F* we have discussed about F^+

the useful info is in the minimal cover of *F*

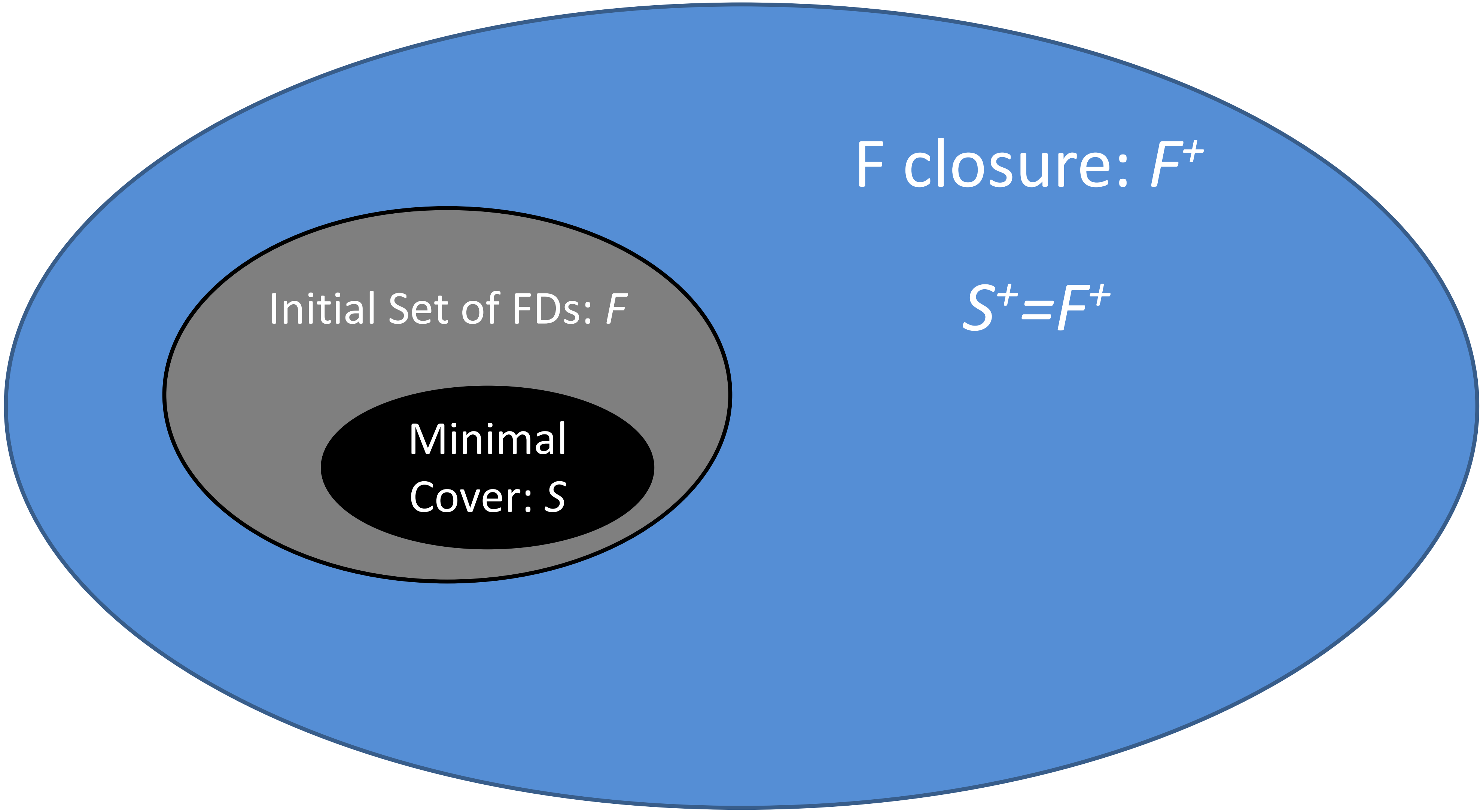
"the smallest subset of FDs S: $S^+ = F^+$"

Formally: minimal cover S for a set of FDs F:

(1) $S^+ = F^+$

(2) RHS of each FD in S is a single attribute

(3) if we remove any FD from S or remove any attribute from its LHS the closure is not F^+



*here, the subset notation is meant as “simpler set of FDs”

Example of Minimal Cover

***R**(C, S, J, D, P, Q, V)*

key C (C^+={C, S, J, D, P, Q, V})

J, P → C

S, D → P

J → S

(1) put FDs in standard form

single attribute on the RHS using decomposition

(2) minimize the LHS

check if by removing attr. equivalence is preserved

(3) delete redundant FDs

<u>Minimal cover:</u>

J, P → C

S, D → P

J → S

~~C → C~~, ~~C → S~~, C → J, C → D, ~~C → P~~, C → Q, C → V

trivial transitivity union & transitivity

Example of Minimal Cover

R*(C, S, J, D, P, Q, V)*

key C (C^+={C, S, J, D, P, Q, V})

J, P → C

S, D → P

J → S

<u>Minimal cover</u>:

J, P → C

S, D → P

J → S

C → J, C → P, C → Q, C → V

(1) put FDs in standard form

single attribute on the RHS using decomposition

(2) minimize the LHS

check if by removing attr. equivalence is preserved

(3) delete redundant FDs

This is useful to decide how to solve the problem of redundancy (decomposition)!

More on that next time!!

Summary

Functional Dependencies and (Super)Keys

Reasoning with FDs:
(1) given a set of FDs, infer all implied FDs
(2) given a set of attributes X, infer all attributes that are functionally determined by X

Next: how to use FDs to detect that a table is "bad"?