CS 6400 A

Database Systems Concepts and Design

Lecture 5
09/04/24

Logistics

Assignment 1 due next Wednesday

- Sep 11 @ 11:59PM

Project proposal

- Groups of 3-4, total of 20 groups
- Sign up under People->Project Groups on canvas
- Use the ”Find teammates” post on Piazza
- Due Oct 2

Design theory for relational databases

There are many ways to design a relational database schema

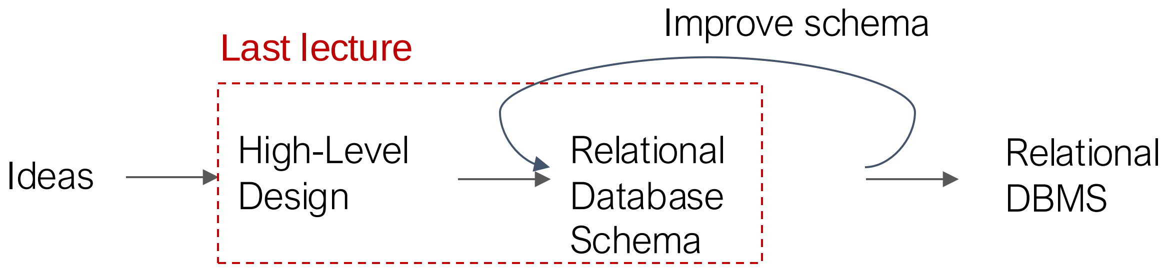
- E.g., we just learned how to use an E/R diagram

It is also common to improve the initial schema (esp. eliminating redundancy)

- Often, the problem is combining too much into one relation

Fortunately, there is a well-developed design theory for good schema design

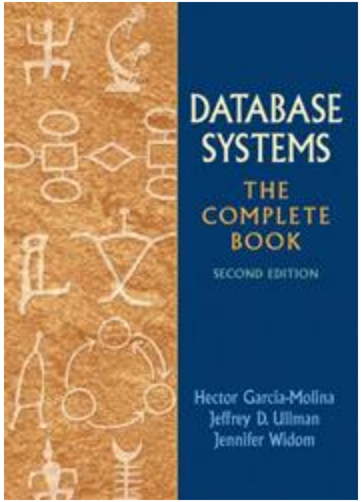
- Functional dependencies, normalization, multivalued dependencies
- One of the reasons Databases are powerful and so widely used



Reading Materials

Database Systems: The Complete Book (2nd edition)

- Chapter 3: Design Theory for Relational Databases (3.1 – 3.5)



Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang and CS145 (Intro to Big Data Systems) taught by Peter Bailis

Agenda

1. Normal forms & functional dependencies

2. Finding functional dependencies

3. Closures, superkeys & keys

1. Normal forms & functional dependencies

Normal Forms

- 1st Normal Form (1NF) = All tables are flat
- 2nd Normal Form = disused
- Boyce-Codd Normal Form (BCNF)
- 3rd Normal Form (3NF)
- 4th and 5th Normal Forms = see textbooks

DB designs based on functional dependencies, intended to prevent data anomalies

Our focus in this lecture + next one

1st Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1st NF

1NF Constraint: Types must be atomic!

Data Anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains **redundant** information!

Data Anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

If we update the room number for one tuple, we get inconsistent data = an **update anomaly**

Data Anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class is in!
= a <u>delete anomaly</u>

Data Anomalies

A poorly designed database causes *anomalies*:

...	CS229	C12

Student	**Course**	**Room**
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Similarly, we can't reserve a room without students = an <u>insert anomaly</u>

Data Anomalies

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Eliminate anomalies by **decomposing relations.**

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

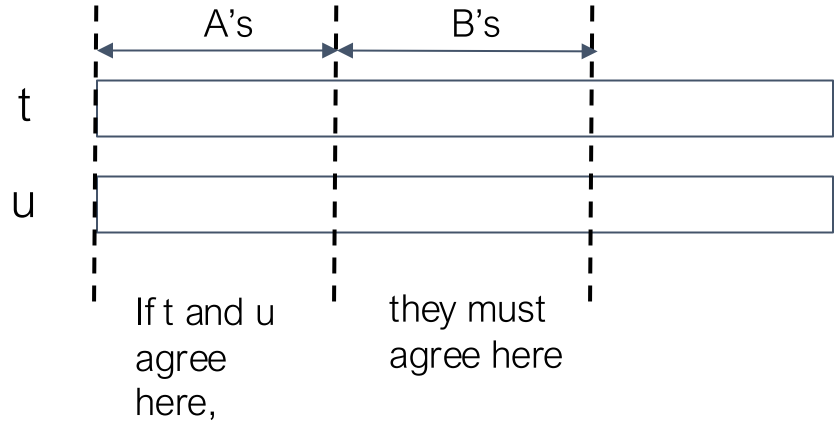
Goal: develop theory to understand why this design may be better and how to find this decomposition…

Functional Dependencies

Functional dependency (FD)

Definition: if two tuples of R agree on all the attributes $A_1, A_2, \ldots, A_n$, they must also agree on (or functionally determine) $B_1, B_2, \ldots, B_m$

- Denoted as $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$



A->B means that "whenever two tuples agree on A then they agree on B."

Splitting/combining rule

- Splitting/combining can be applied to the right sides of FD's

$$A_1A_2 \dots A_n \rightarrow B_1B_2 \dots B_m$$

Splitting rule ↓ ↑ Combining rule

$$A_1A_2 \dots A_n \rightarrow B_1, \quad A_1A_2 \dots A_n \rightarrow B_2, \quad \dots, \quad A_1A_2 \dots A_n \rightarrow B_m$$

Splitting/combining rule

- For example,

title year → length genre studioName

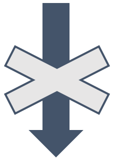
‖

title year → length
title year → genre
title year → studioName

Splitting rule

- Splitting rule does not apply to the left sides of FD's

title year → length



title → length
year → length

Functional Dependencies as Constraints

A functional dependency is a form of constraint

- Holds on some instances (but not others) – can check whether there are violations
- Part of the schema, helps define a valid instance

Recall: an instance of a schema is a multiset of tuples conforming to that schema, i.e. a table

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Note: The FD {Course} -> {Room} holds on this instance

Functional Dependencies as Constraints

Note that:

- You can check if an FD is **violated** by examining a single instance;
- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  - This would require checking every valid instance

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

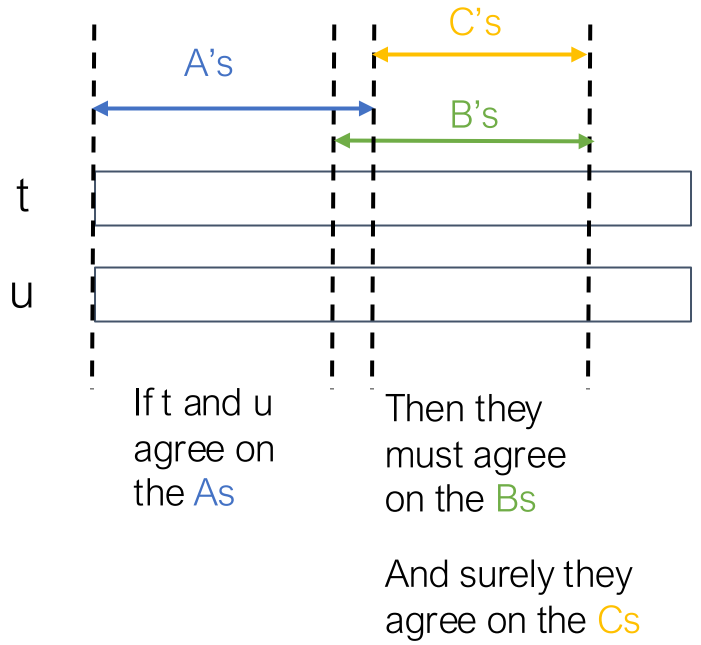
However, cannot prove that the FD {Course} -> {Room} is part of the schema

Trivial functional dependencies

A constraint is *trivial* if it holds for every possible instance of the relation.

Trivial FDs:
$A_1A_2 \ldots A_n \rightarrow B_1\ B_2 \ldots B_m$ such that $\{B_1, B_2, \ldots B_m\} \subseteq \{A_1,A_2, \ldots ,A_n\}$

Trivial dependency rule:
$A_1A_2 \ldots A_n \rightarrow B_1\ B_2 \ldots B_m$ is equivalent to $A_1A_2 \ldots A_n \rightarrow C_1\ C_2 \ldots C_k$ where the C's are the B's that are not also A's



Exercise #1

Q1: Find an FD that holds on this instance
Q2: Find an FD that is violated on this instance

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

2. Finding functional dependencies

FDs for Relational Schema Design

High-level idea: why do we care about FDs?

1. Start with some relational schema
2. Find out its functional dependencies (FDs)
3. Use these to design a better schema
    1. One which minimizes possibility of anomalies

This part can be tricky!

Finding Functional Dependencies

There can be a large number of FDs…

Let's start with this problem:
Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Three simple rules called **Armstrong's Rules**.

1. Reflexivity,
2. Augmentation,
3. Transitivity

Armstrong's axioms

You can derive any FDs that follows from a given set using these axioms:

1. **Reflexivity**:
   If Y is a subset of X, then X $\rightarrow$ Y

This means that a set of attributes always determines a subset of itself

2. **Augmentation**:
   If X $\rightarrow$ Y, then XZ $\rightarrow$ YZ for any Z

This means we can add the same attributes to both sides of a functional dependency.

3. **Transitivity**:
   If X $\rightarrow$ Y and Y $\rightarrow$ Z, then X $\rightarrow$ Z

This allows us to chain functional dependencies.

Armstrong's axioms

- Does AB → D follow from the FDs below?

AB → C

BC → AD

D → E

CF → B

1. AB → C (given)
2. BC → AD (given)

Armstrong’s axioms

- Does AB → D follow from the FDs below?

AB → C
BC → AD
D → E
CF → B

1. AB → C (given)
2. BC → AD (given)
3. AB → BC (Augmentation on 1)

Armstrong's axioms

- Does AB → D follow from the FDs below?

AB → C

BC → AD

D → E

CF → B

1. AB → C (given)
2. BC → AD (given)
3. AB → BC (Augmentation on 1)
4. AB → AD (Transitivity on 2,3)

Armstrong’s axioms

- Does AB → D follow from the FDs below?

AB → C

BC → AD

D → E

CF → B

1. AB → C (given)
2. BC → AD (given)
3. AB → BC (Augmentation on 1)
4. AB → AD (Transitivity on 2,3)
5. AD → D (Reflexivity)

Armstrong's axioms

- Does AB → D follow from the FDs below?

AB → C

BC → AD

D → E

CF → B

1. AB → C (given)
2. BC → AD (given)
3. AB → BC (Augmentation on 1)
4. AB → AD (Transitivity on 2,3)
5. AD → D (Reflexivity)
6. AB → D (Transitivity on 4,5)

Can we find an algorithmic way to do this?

Closures

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the closure, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}+$

A, B

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}^+$

A, B, C

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}+$

A, B, C, D

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the closure, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}^+$

A, B, C, D, E

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}^+$

A, B, C, D, E

Cannot be expanded further, so this is a closure

Closure algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; **do**:

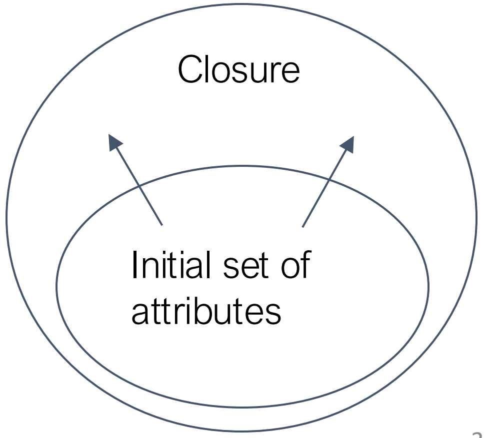
if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by F

and $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X.

Return X as X^+

Helps to split the FD's of F so each FD has a single attribute on the right



The algorithm

- only produces true FDs
- Discovers all true FDs

3. Closures, Superkeys & Keys

Why Do We Need the Closure?

With closure we can find all FD's easily

To check if $X \rightarrow A$

1. Compute X^+
2. Check if $A \in X^+$

Note here that **X** is a set of attributes, but **A** is a single attribute. Why does considering FDs of this form suffice?

Recall the split/combine rule:
$X \rightarrow A_1, \ldots, X \rightarrow A_n$
implies
$X \rightarrow \{A_1, \ldots, A_n\}$

Using Closure to Infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

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

Using Closure to Infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

Using Closure to Infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

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

Y is in the closure of X

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$, $\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$, $\{A,C,D\} \rightarrow \{B\}$

The FD $X \rightarrow Y$ is non-trivial

Keys and Superkeys

A superkey is a set of attributes $A_1, \ldots, A_n$ s.t.
for any other attribute B in R,
we have $\{A_1, \ldots, A_n\} \rightarrow B$

i.e. all attributes are functionally determined by a superkey

A key is a minimal superkey

This means that no subset of a key is also a superkey
(i.e., dropping any attribute from the key makes it no longer a superkey)

Keys and Superkeys

Q: What are superkeys and keys in the following relation?

{title, year, length, starName} is a superkey

{title, year, starName} is a key
{title, year} is not a key because title year → starName is not an FD
{year, starName} is not a key because year starName → title is not an FD
{title, starName} is not a key because title starName → year is not an FD

title	**year**	**length**	**genre**	**studioName**	**starName**
Ponyo	2008	103	anime	Ghibli	Yuria Nara
Ponyo	2008	103	anime	Ghibli	Hiroki Doi
Oldboy	2003	120	mystery	Show East	Choi Min-Sik

Finding Keys and Superkeys

For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Example of Finding Keys

Product(name, price, category, color)

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

What is a key?

Example of Finding Keys

Product(name, price, category, color)

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

$\{name, category\}^+$ = {name, price, category, color}
= the set of all attributes
⟹ this is a **superkey**
⟹ this is a **key**, since neither name nor category alone is a superkey

Exercise #2

Given R(A, B, C, D) and FD's AB $\rightarrow$ C, C $\rightarrow$ D, D $\rightarrow$ A

- What are all keys of R?