Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#16: Schema Refinement & Normalization

Administrivia

- HW5 is due today.
- HW6 is due Tuesday March 24th.

Database Design

- How do we design a "good" database?
  - Want to ensure the integrity of the data.
  - Want to get good performance.

This Week

Next Week

- Relational DBMS vs. NoSQL DBMS

Example

Students(studentId, courseId, room, grade, name, address**)**

studentId	courseId	room	grade	name	address
123	15-415	GHC 6115	A	Christos	Pittsburgh
456	15-721	GHC 8102	B	Tupac	Los Angeles
789	15-415	GHC 6115	A	Obama	Chicago
012	15-415	GHC 6115	A	Waka Flocka	Atlanta

Redundancy Problems

- Update Anomalies
  - *If the room number changes, we need to make sure that we change all students records.*
- Insert Anomalies
  - *May not be possible to add a student unless they’re enrolled in a course.*
- Delete Anomalies
  - *If all the students enrolled in a course are deleted, then we lose the room number.*

Example

STUDENTS

studentId	name	address
123	Christos	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

COURSES

studentId	courseId	grade
123	15-415	A
456	15-721	B
789	15-415	A
012	15-415	A

ROOMS

courseId	room
15-415	GHC 6115
15-721	GHC 8102

This Week: why this decomposition is better and how to find it.

Today’s Class

- Motivation
- Functional Dependencies
- Armstrong’s Axioms
- Closures
- Canonical Cover

Functional Dependencies

- A form of a **constraint**:
  - Part of the schema to define a valid instance.
- Definition: $\mathbf{X} \rightarrow \mathbf{Y}$
  - ***The value of ‘X’ functionally defines the value of ‘Y’.***

Functional Dependencies

- Formal Definition:
  - $X \rightarrow Y \Rightarrow (t_1[x] = t_2[x] \Rightarrow t_1[y] = t_2[y])$
  - ***If two tuples agree on the ‘X’ attribute, then they must agree on the ‘Y’ attribute too.***

studentId	name	address
123	Christos	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

studentId→name

Functional Dependencies

- FD is a constraint, that it says that it allows instances for which where the FD holds.
- You can check if an FD is violated by an instance, but cannot prove that an FD is part of the schema using an instance.

studentId	name	address
123	Christos	Pittsburgh
456	Tupac	Los Angeles
789	Obama	Chicago
012	Waka Flocka	Atlanta

studentId→name

?**address→name**?

Functional Dependencies

- Note that the two FDs **X→Y** and **X→Z** can be written in shorthand as **X→YZ**.
- But **XY→Z** is ***not*** the same as the two FDs **X→Z** and **Y→Z**.

Defining FDs in SQL

```
CREATE ASSERTION student-name
 CHECK (NOT EXISTS
  (SELECT * FROM students AS s1,
                 students AS s2
    WHERE s1.studentId = s2.studentId
      AND s1.name <> s2.name))
```

FD: studentId → name

Make sure that no two students ever have
the same id without the same name.

Combining FDs in SQL

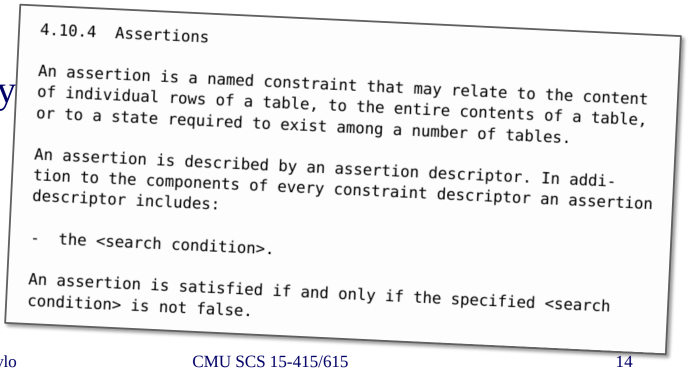
```
CREATE ASSERTION student-name-address
 CHECK (NOT EXISTS
  (SELECT * FROM students AS s1,
                 students AS s2
    WHERE s1.studentId = s2.studentId
      AND ((s1.name <> s2.name
       OR  (s1.address <> s2.address)))
```

FD$_1$: studentId → name
FD$_2$: studentId → address

Make sure that no two students ever have the same id without the same name and address.

SQL Assertions

- ***WARNING: No major DBMS supports SQL-92 assertions.***
- Why



4.10.4 Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

An assertion is described by an assertion descriptor. In addition to the components of every constraint descriptor an assertion descriptor includes:

- the <search condition>.

An assertion is satisfied if and only if the specified <search condition> is not false.

Defining FDs in IBM DB2

```
CREATE TABLE students (
   studentId INT PRIMARY KEY,
   name VARCHAR(32),
   ⋮
   CONSTRAINT student_name
        CHECK (name)
DETERMINED BY (studentId) );
```

FD: studentId → name

http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.sql.ref.doc/doc/r0000927.html?cp=SSEPGG_9.7.0%2F2-10-6-90&lang=en

Why Should I Care?

- FDs seem important, but what can we actually do with them?
- They allow us to decide whether a database design is **correct**.
  - Note that this different then the question of whether it’s a good idea for performance…

Implied Dependencies

Students(studentId, courseId, grade, name, address**)**

studentId	courseId	grade	name	address
123	15-415	A	Christos	Pittsburgh
456	15-721	B	Tupac	Los Angeles
789	15-415	A	Obama	Chicago
012	15-415	A		

These holds for any instance!

Provided FDs

studentId → name, address
studentId, courseId → grade

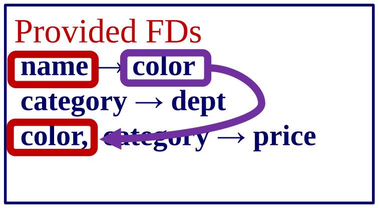
Implied FDs

studentId, courseId →
grade, name, address
studentId, courseId →
studentId

Another Example

Product(name, color, category, dept, price**)**

name	color	category	dept	price
Gizmo	Green	Gadget	Toys	9.99
Widget	Black	Gadget	Toys	49.99
Gizmo	Green	Squirrels	Garden	19.99



Provided FDs

name → color
category → dept
color, category → price

Implied FDs

name, category → price

Implied Dependencies

- **Q:** Given a set of FDs **{f_1, … f_n}**, how do we decide whether FD **g** holds?

- **A:** Compute the ***closure*** using Armstrong's Axioms (chapter 19.3):
  - Reflexivity
  - Augmentation
  - Transitivity

The set of all implied FDs

Armstrong's Axioms – Reflexivity

- If **X ⊇ Y**, then **X→Y**.
- Example: studentId, name → studentId

Armstrong’s Axioms – Augmentation

- If $\mathbf{X \rightarrow Y}$, then $\mathbf{XZ \rightarrow YZ}$ for any $\mathbf{Z}$.
- Example: If studentId → name,
  then studentId, grade → name, grade

Armstrong’s Axioms – Transitivity

- If $\mathbf{X \rightarrow Y}$ and $\mathbf{Y \rightarrow Z}$, then $\mathbf{X \rightarrow Z}$.
- Example: If studentId→address and
  address →taxRate, then studentId→taxRate

Armstrong’s Axioms

- **Reflexivity**:
  - $X \supseteq Y \Rightarrow X \rightarrow Y$
- **Augmentation**:
  - $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- **Transitivity**:
  - $(X \rightarrow Y) \wedge (Y \rightarrow Z) \Rightarrow X \rightarrow Z$

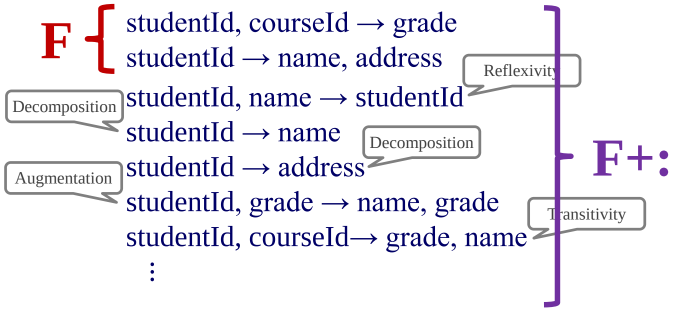
Additional Rules

- **Union**:
  - $(X \rightarrow Y) \wedge (X \rightarrow Z) \Rightarrow X \rightarrow YZ$
- **Decomposition:**
  - $X \rightarrow YZ \Rightarrow (X \rightarrow Y) \wedge (X \rightarrow Z)$
- **Pseudo-transitivity:**
  - $(X \rightarrow Y) \wedge (YW \rightarrow Z) \Rightarrow XW \rightarrow Z$

Closures

- Given a set **F** of FDs **{f_1, … f_n}**, we define the closure **F+** is the set of all implied FDs.

Students(studentId, courseId, grade, name, address**)**

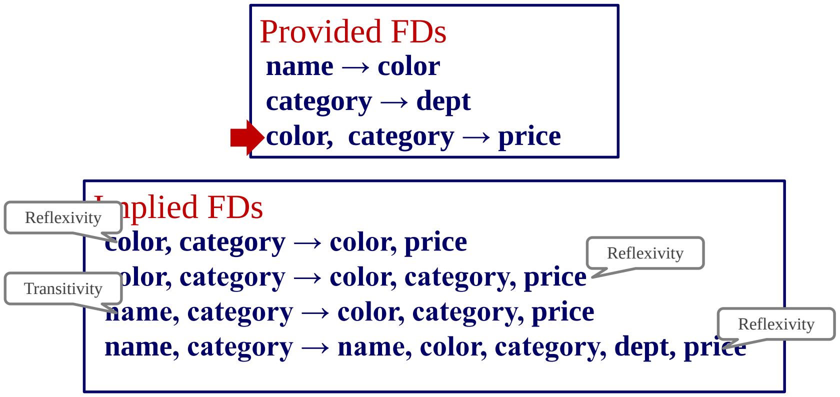


F {
studentId, courseId → grade
studentId → name, address

studentId, name → studentId
studentId → name
studentId → address
studentId, grade → name, grade
studentId, courseId→ grade, name
⋮

Another Example

Product(name, color, category, dept, price**)**



Provided FDs
name → color
category → dept
color, category → price

Implied FDs
color, category → color, price
color, category → color, category, price
name, category → color, category, price
name, category → name, color, category, dept, price

Why Do We Need the Closure?

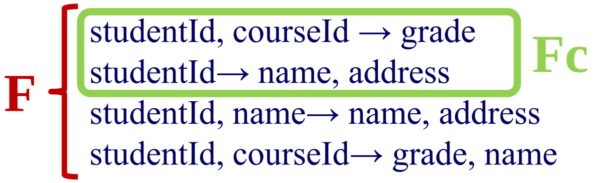
- With closure we can find all FD’s easily.
- We can then compute the **attribute closure**
  - For a given attribute X, the attribute closure X+ is the set of all attributes such that X→A can be inferred using the Armstron Axioms.
- To check if **X→A**,
  - Compute **X+**
  - Check if **A ∈ X+**

But Again, Why Should I Care?

- Maintaining the closure at runtime is expensive:
  - The DBMS has to check all the constraints for every insert, update, delete operation.
- We want a **minimal set of FDs** that was enough to ensure correctness.

Canonical Cover

- Given a set **F** of FDs **{f_1, … f_n}**, we define the closure **Fc** is the minimal set of all FDs.



studentId, courseId → grade
studentId→ name, address
studentId, name→ name, address
studentId, courseId→ grade, name

Canonical Cover Definition

- Three properties for the canonical cover **Fc**:
  1. The RHS of every FD is a single attribute.
  2. The closure of **Fc** is identical to the closure of **F** (i.e., **Fc** = **F** are equivalent).
  3. The **Fc** is minimal (i.e., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated.

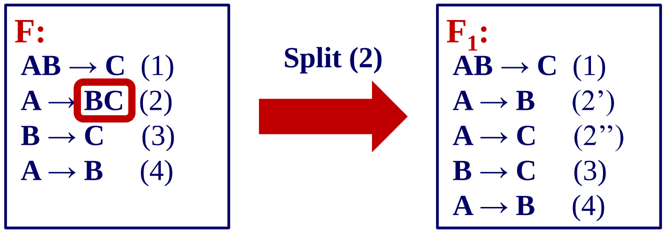
Canonical Cover Definition

- For #3, we need to eliminate all extraneous attributes from our set of FDs.
  - An attribute is “extraneous” if the closure is the same, before and after its elimination.

Computing the Canonical Cover

- Given a set **F** of FDs, 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

Computing the Canonical Cover



F:

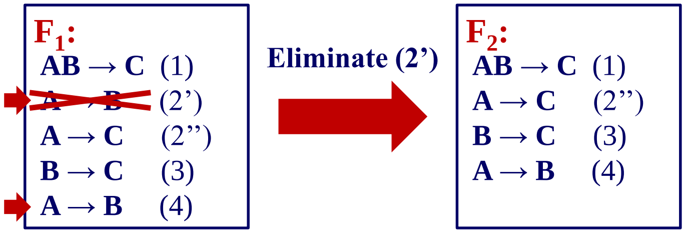
AB → C	(1)
A → BC	(2)
B → C	(3)
A → B	(4)

Split (2)

F_1:

AB → C	(1)
A → B	(2')
A → C	(2'')
B → C	(3)
A → B	(4)

Computing the Canonical Cover



F_1:

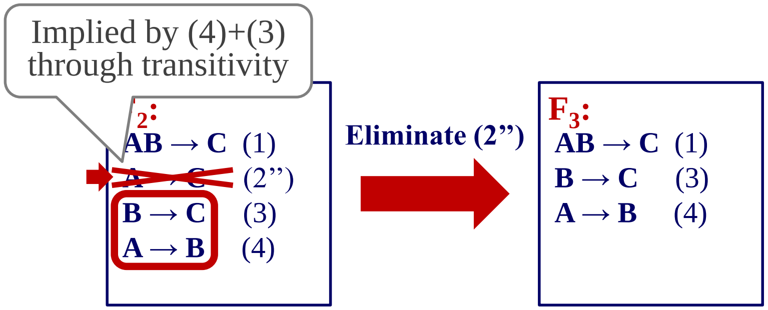
AB → C	(1)
~~**A → B**~~	(2')
A → C	(2'')
B → C	(3)
A → B	(4)

Eliminate (2')

F_2:

AB → C	(1)
A → C	(2'')
B → C	(3)
A → B	(4)

Computing the Canonical Cover



Implied by (4)+(3) through transitivity

F_2:

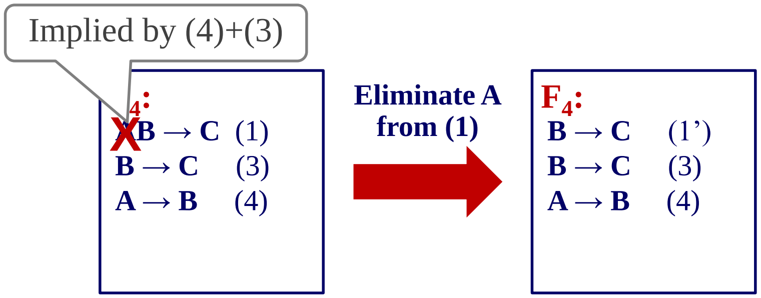
AB → C	(1)
~~**A → C**~~	(2'')
B → C	(3)
A → B	(4)

Eliminate (2'')

F_3:

AB → C	(1)
B → C	(3)
A → B	(4)

Computing the Canonical Cover



Implied by (4)+(3)

F_4:

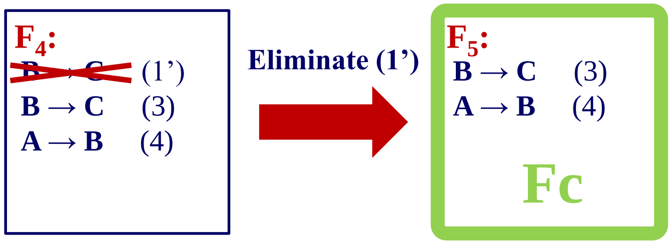
~~A~~B → C	(1)
B → C	(3)
A → B	(4)

Eliminate A from (1)

F_4:

B → C	(1')
B → C	(3)
A → B	(4)

Computing the Canonical Cover



F$_4$:

~~B → C~~ (1’)

B → C (3)

A → B (4)

Eliminate (1’)

F$_5$:

B → C (3)

A → B (4)

Fc

✓ Nothing is extraneous

✓ All RHS are single attributes

✓ Final & original set of FDs are equivalent (same closure)

No Really, Why Should I Care?

- The canonical cover is the minimum number of assertions that we need to implement to make sure that our database integrity is correct.
- Allows us to find the **super key** for a relation.

Relational Model: Keys

- **Super Key**:
  – Any set of attributes in a relation that functionally determines all attributes in the relation.
- **Candidate Key**:
  – Any super key such that the removal of any attribute leaves a set that does not functionally determine all attributes.

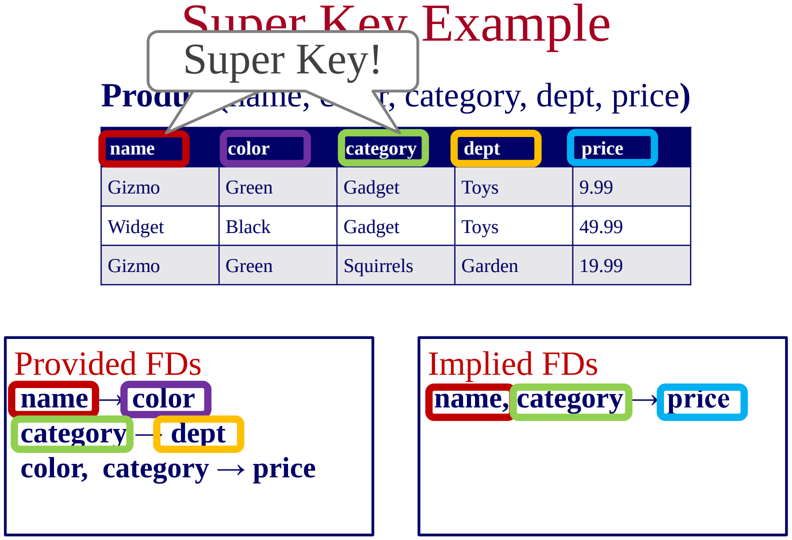
Relational Model: Keys

- **Super Key**:
  – Set of fields for which there are no two distinct tuples that have the same values for the attributes in this set.
- **Candidate Key**:
  – Set of fields that uniquely identifies a tuple according to a key constraint.

But Why Care About Super Keys?

- It is going to help us determine whether it’s okay to split a table into multiple sub-tables.
- Super keys ensure that we are able to recreate the original relation through joins.

Super Key Example



Super Key!

Product(name, color, category, dept, price)

name	color	category	dept	price
Gizmo	Green	Gadget	Toys	9.99
Widget	Black	Gadget	Toys	49.99
Gizmo	Green	Squirrels	Garden	19.99

Provided FDs

name → color
category → dept
color, category → price

Implied FDs

name, category → price

Summary

- How do we guarantee that F = F’?
  - Closures
- How do we find a minimal F’ for F?
  - Canonical Cover