Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)

Functional Dependencies

- Difference between holding on an *instance* and holding on *all legal relations*

Title	Year	Length	inColor	StudioName	prodC#	StarName
Star wars	1977	121	Yes	Fox	128	Hamill
Star wars	1977	121	Yes	Fox	128	Fisher
Star wars	1977	121	Yes	Fox	128	H. Ford
King Kong	1933	100	no	RKO	20	Fay

- *Title → Year holds on this instance*
- *Is this a true functional dependency ? **No**.*
  - *Two movies in different years can have the same name.*
- Can't draw conclusions based on a *single instance*
  - Need ***domain knowledge to decide which FDs hold***

FDs and Redundancy

- Consider a table: R(A, B, C):
  - With FDs: B → C, and A → BC
  - So "A" is a Key, but "B" is not
- So: there is a FD whose left hand side is not a key
  - **Leads to redundancy**

Since B is not unique, it may be duplicated
Every time B is duplicated, so is C

Not a problem with A → BC
A can never be duplicated

A	B	C
a1	b1	c1
a2	b1	c1
a3	b1	c1
a4	b2	c2
a5	b2	c2
a6	b3	c3
a7	b4	c1

Not a duplication → Two different tuples just happen to have the same value for C

Functional Dependencies

- Functional dependencies and *keys:*
  - A *key constraint* is a specific form of a FD.
  - E.g. if α is a superkey for *R,* then:
- $\alpha \rightarrow R$
  - Similarly for *candidate keys and primary keys.*

- Deriving FDs
  - A set of FDs may imply other FDs
  - e.g. If *A → B, and B → C*, then clearly *A → C*
  - We will see a formal method for inferring this later

Definitions

- 1. A relation instance ***r satisfies*** a set of functional dependencies, ***F***, if the FDs ***hold*** on that relation
- ***2. F holds on*** a relation schema ***R*** if no legal (allowable) relation instance of ***R*** violates it
- 3. A functional dependency, $\alpha \rightarrow \beta$, is called ***trivial*** if:
  - α is a superset of β
  - e.g. MovieName, length → length
- 4. Given a set of functional dependencies, ***F,*** its ***closure,***

  F^+ is all the FDs that are implied by FDs in ***F.***

Approach

- 1. We will encode and list all our knowledge about the schema
  - Functional dependencies (FDs)
  - Also:
    - Multi-valued dependencies (briefly discuss later)
    - Join dependencies etc…
- 2. We will define a set of rules that the schema must follow to be considered good
  - "Normal forms": 1NF, 2NF, BCNF, 3NF, 4NF, …
  - A normal form specifies constraints on the schemas and FDs
- 3. If not in a "normal form", we modify the schema

BCNF: Boyce-Codd Normal Form

- A relation schema ***R*** is "in BCNF" if:
  - Every functional dependency $\alpha \rightarrow \beta$ that holds on it is ***EITHER***:
    - 1. Trivial ***OR***
    - 2. α is a superkey of R

- ***Why is BCNF good ?***
  - Guarantees ***no redundancy*** because of a functional dependency

  - Consider a relation ***r(A, B, C, D)*** with functional dependency
    A→B and two tuples: (a1, b1, c1, d1), and (a1, b1, c2, d2)
  - ***b1*** is repeated because of the functional dependency

  - BUT this relation is not in BCNF
    A→B is neither trivial nor is ***A*** a superkey for the relation

BCNF and Redundancy

- ***Why does redundancy arise ?***
  - Given a FD, $\alpha \rightarrow \beta$, if α is repeated ($\beta - \alpha$) has to be repeated
  1. If rule 1 is satisfied, ($\beta - \alpha$) is empty, so not a problem.
  2. If rule 2 is satisfied, then α can't be repeated, so this doesn't happen either

- Hence no redundancy because of FDs in BCNF
  - Redundancy may exist because of other types of dependencies
    - Higher normal forms used for that (specifically, 4NF)
  - Data may naturally have duplicated/redundant data
    - We can't control that unless a FD or some other dependency is defined

Approach

- 1. We will encode and list all our knowledge about the schema:
  - Functional dependencies (FDs); Multi-valued dependencies; Join dependencies etc…
- 2. We will define rules the schema must follow to be "good"
  - "Normal forms": 1NF, 2NF, 3NF, BCNF, 4NF, …
  - A normal form specifies constraints on the schemas and FDs
- 3. If not in a "normal form", we modify the schema
  - Through lossless decomposition (splitting)
  - Or direct construction using the dependencies information

BCNF

r(A, B, C, D) with A → B and:
(a1, b1, c1, d1), and (a1, b1, c2, d2)

- What if the schema is not in BCNF ?
  - *Decompose (split) the schema into two pieces.*

R_1	R_2
(a1, b1)	(a1, c1, d1) (a1, c2, d2)

- From the previous example: split the schema into:
  - *r1(A, B), r2(A, C, D)*
  - The first schema is in BCNF, the second one may not be (and may require further decomposition)
  - No repetition now: ***r1*** contains ***(a1, b1)***, but ***b1*** will not be repeated
- Careful: you want the decomposition to be lossless
  - *No information should be lost*
    - The above decomposition is lossless
  - We will define this more formally later

Normalization

BCNF

- Recall that ***R*** is in BCNF if every FD, $\alpha \rightarrow \beta$, is either:
  - 1. Trivial, or
  - *2.* α is a *superkey* of *R*
- *No redundancy*

- What if the schema is not in BCNF ?
  - *Decompose (split) the schema into two pieces.*
  - Careful: you want the decomposition to be lossless

Achieving BCNF Schemas

- For all dependencies $\alpha \rightarrow \beta$ in F^+, check if α is a superkey
  - (attribute closure)

- If not, then
  - Choose a dependency in F^+ that breaks the BCNF rules, say $\alpha \rightarrow \beta$
  - Create $R_1 = \alpha\beta$
  - Create $R_2 = R - (\beta - \alpha)$.
  - Note that: $R_1 \cap R_2 = \alpha$ and $\alpha \rightarrow \alpha\beta$, so:
    - α is a *superkey* of R_1
    - *lossless decomposition* (lossless if intersection of two attribute sets is key for one)
- Repeat for R_1, *and* R_2
  - Define F_i to be all dependencies in F^+ that contain only attributes in R_i

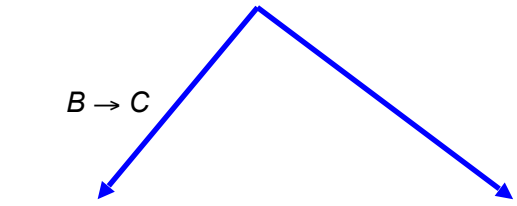
Note:
$(R - (\beta - \alpha)) == (R - \beta)$
if no extraneous attributions in FDs

We use **$(R - \beta)$** *in this course.*

Achieving BCNF Schemas

Example 1

R = (A, B, C)
F = {$A \rightarrow B$, $B \rightarrow C$}
Candidate keys = {A}
BCNF? No. $B \rightarrow C$ violates.



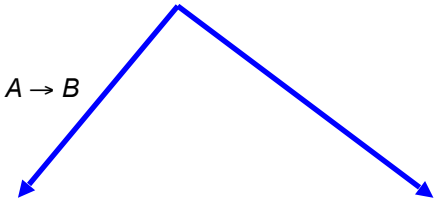
R1 = (B, C)
F1 = {$B \rightarrow C$}
Candidate keys = {B}
BCNF = true

R2 = (A, B)
F2 = {$A \rightarrow B$}
Candidate keys = {A}
BCNF = true

- Dependency-preserving?
  - yes

Example 2a

R = (A, B, C, D, E)
F = {$A \rightarrow B$, $BC \rightarrow D$}
Candidate keys = {ACE}
BCNF = Violated by {$A \rightarrow B$, $BC \rightarrow D$}



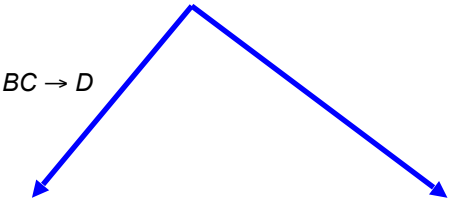
R1 = (A, B)
F1 = {$A \rightarrow B$}
Candidate keys = {A}
BCNF = true

R2 = (A, C, D, E)
F2 = {}
Candidate keys = {ACDE}
BCNF = true

- Dependency-preserving?
  - no: lost $BC \rightarrow D$

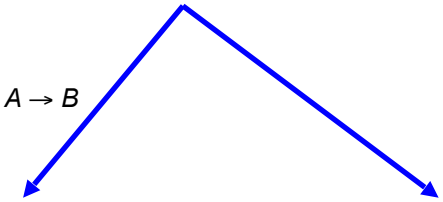
Example 2b

R = (A, B, C, D, E)
F = {$A \rightarrow B$, $BC \rightarrow D$}
Candidate keys = {ACE}
BCNF = Violated by {$A \rightarrow B$, $BC \rightarrow D$}



R1 = (BCD)
F1 = {BC → D}
Candidate keys = {BC}
BCNF = true

R2 = (A, B, C, E)
F2 = {$A \rightarrow B$}
Candidate keys = {ACE}
BCNF = false ($A \rightarrow B$)



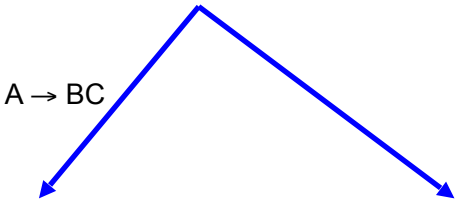
R3 = (A, B)
F3 = {$A \rightarrow B$}
Candidate keys = {A}
BCNF = true

R4 = (A, C, E)
F4 = {} [[only trivial]]
Candidate keys = {ACE}
BCNF = true

- Dependency-preserving?
  - yes

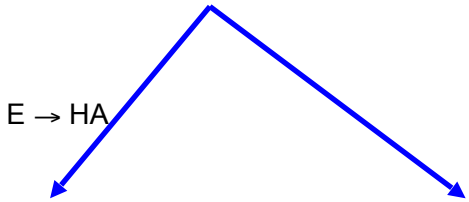
Example 3

R = (A, B, C, D, E, H)
F = {A → BC, E → HA}
Candidate keys = {DE}
BCNF = Violated by {A → BC} and {E → HA}



R1 = (A, B, C)
F1 = {A → BC}
Candidate keys = {A}
BCNF = true

R2 = (A, D, E, H)
F2 = {E → HA}
Candidate keys = {DE}
BCNF = false (E → HA)



R3 = (E, H, A)
F3 = {E → HA}
Candidate keys = {E}
BCNF = true

R4 = (ED)
F4 = {} [[only trivial]]
Candidate keys = {DE}
BCNF = true

- Dependency-preserving?
  - yes