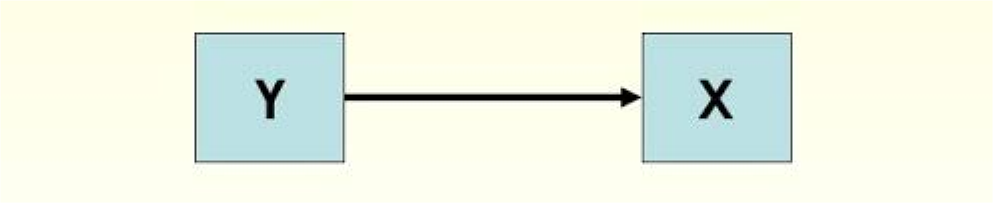
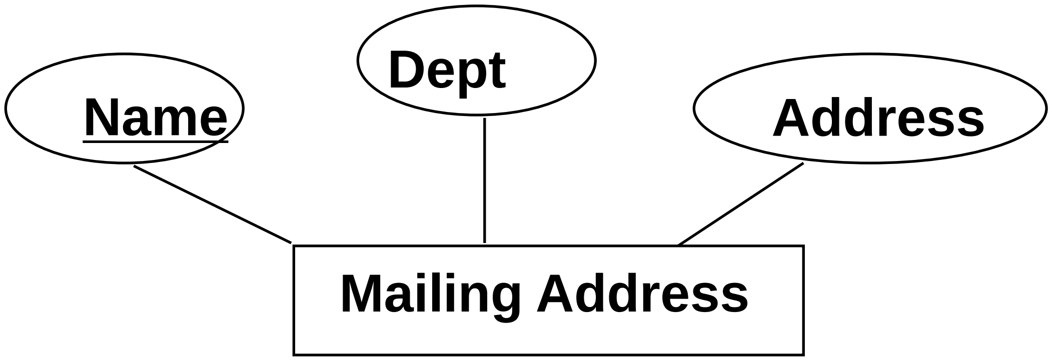
Functional Dependencies

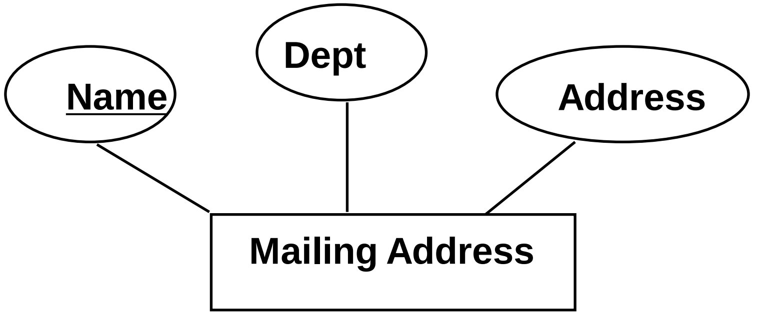
R&G Chapter 19



Imagine that we've created a perfectly good entity for mailing addresses at Stevens



What would an instance look like?



Name	Dept	Address
Alice	CS	Gateway South
Bob	CS	Gateway South
Carol	CS	Gateway South
David	ECE	Burchard

Mailing Address Table

Observation: the same departments are always associated with the same address

BAD database design: as it contains **DATA REDUNDANCY**!

The Evils of Redundancy

- Redundancy
  - Some information is stored repeatedly in the database
  - The ROOT of several problems associated with bad relational schema design

Example of Redundancy

- **Consider the following relation and its instance: Person(<u>SSN</u>, Name, Address, <u>Hobby</u>)**

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	123 Main street	Cooking
22345678	Bob	456 Main street	Video game
32345678	Carol	456 Main street	Gardening

- **NOTE: some information is redundant in the instance**

What problems arise because of redundancy?

- Redundancy gives rise to ANOMALIES
  - **UPDATE ANOMALIES**
  - **INSERTION ANOMALIES**
  - **DELETION ANOMALIES**

Update Anomaly

If Alan moves to a new place (254 Main street), but only one tuple is updated?

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	254 Main street	Cooking
22345678	Bob	456 Main street	Video game
32345678	Carol	456 Main street	Gardening

Data inconsistency problem!

Insertion Anomaly

A new hobby tuple of Alan is inserted, but with a different address...?

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	123 Main street	Cooking
12345678	Alan	134 Main street	programming
22345678	Bob	456 Main street	Video game
32345678	Carol	456 Main street	Gardening

Data inconsistency problem (again)!

Deletion Anomaly

Delete Bob's hobby by deleting his tuple

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	123 Main street	Cooking
22345678	Bob	456 Main street	Video game
32345678	Carol	456 Main street	Gardening

Bob's address info will not exist in the db!

Okay, data redundancy is bad.

But how do I know if each person only has one address ?

Functional Dependency
Informal Definition

- A set of attributes can determine another set of attributes through a ***functional dependency (FD)***.
  - So if SSN determines Address, we say that there's a *functional dependency*: Address depends on SSN.

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	123 Main street	Cooking
22345678	Bob	456 Main street	Video game
22345678	Bob	456 Main street	drinking

Functional Dependency (FD)
Formal Definition

- A functional dependency X -> Y holds over relation schema R if the following holds:
  - For any two records t, t’ in R, if t[X]=t’[X], then t[Y]=t’[Y].
    - i.e., all records that have the same X values always have the same Y values (but not vice versa)

- Can read “->” as “determines”

FD Example

SSN	Name	Address	Hobby
12345678	Alan	123 Main street	Reading
12345678	Alan	123 Main street	Cooking
22345678	Bob	456 Main street	Video game
44444444	Carol	456 Main street	Gardening

- FD: SSN -> Address (i.e., any two records of the same SSN must have the same address)
  - Since SSN is the key, the FD implies that each person only has one address
- Do we have: Address -> SSN?

X-> Y does not imply that Y -> X!

Why FDs can help to solve data redundancy problem?

Example: Constraints on Entity Set

- **Consider relation of table** Hourly_Emps:

  Hourly_Emps (*ssn*, *name*, *lot*, *rating*, *wage_per_hr*, *hrs_per_wk*)

  – We sometimes denote a relation schema by listing the attributes: e.g., SNLRWH

What are the possible FDs on Hourly_Emps?

ssn is the key: S -> SNLRWH

rating determines *wage_per_hr*: R -> W

lot determines *lot*: L -> L ("trivial" dependnency)

Problems Due to R -> W

SSN	Name	Parking Lot	Rating	Wage	Hour
123-22-3666	Andy	48	22	10	40
231-31-5368	Jess	22	24	10	30
131-24-3650	Andrew	35	5	7	32
434-26-3751	Guldu	2	5	7	40

Hourly_Emps

FDs:
S ->SNLRWH
R -> W

- *Update anomaly*: Can we modify Wage in the 3rd tuple only (but no change on Wage in the 4th tuple)?
- *Insertion anomaly*: What if we want to insert an employee but don't know the hourly wage for his/her rating? (or we get it wrong?)
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

The same 3 problems by data redundancy!

Eliminating Redundancy by Decomposition

- Redundancy can be removed by “chopping” the relation into smaller tables.
- FD’s are used to drive this process.

R->W is causing the problems, so decompose SNLRWH into SNLRH and RW.

SSN	Name	Parking Lot	Rating	Wage	Hour
123-22-3666	Andy	48	22	10	40
231-31-5368	Jess	22	24	10	30
131-24-3650	Andrew	35	5	7	32
434-26-3751	Guldu	2	5	7	40

SSN	Name	Parking Lot	Rating	Hour
123-22-3666	Andy	48	22	40
231-31-5368	Jess	22	24	30
131-24-3650	Andrew	35	5	32
434-26-3751	Guldu	2	5	40

Rating	Wage
22	10
24	10
5	7

R->W

Are the 3 problems solved now?

SNPRH table

SSN	Name	Parking Lot	Rating	Hour
123-22-3666	Andy	48	22	40
231-31-5368	Jess	22	24	30
131-24-3650	Andrew	35	5	32
434-26-3751	Guldu	2	5	40

+

RW table

Rating	Wage
22	10
24	10
5	7

- *Update anomaly*: What if we want to modify Wage of the 3rd tuple?
  - Wage is updated in RW table (including the wage of the 4th tuple).
- *Insertion anomaly*: What if we want to insert an employee but don't know the hourly wage for his/her rating?
  - If the rating exists in RW table, we only need to ensure to insert the rating of the employee in SNPRH table.
- *Deletion anomaly*: What if we delete all employees with rating 5?
  - The wage information of rating 5 still exists in RW table.

Why do we only consider R->W?

SSN	Name	Parking Lot	Rating	Wage	Hour
123-22-3666	Andy	48	22	10	40
231-31-5368	Jess	22	24	10	30
131-24-3650	Andrew	35	5	7	32
434-26-3751	Guldu	2	5	7	40

Hourly_Emps

FDs:
S ->SNLRWH
R -> W

S ->SNLRWH infers that S-> W

Question:
Why R->W is problematic, but S ->W is not?
(Hint: is there any redundancy on (S, W) pairs)?

FDs are NOT just for solving redundancy problem

FDs are NOT just for solving redundancy problem

- Use FDs to infer new FDs
- Use FDs to determine keys

Use FDs to Infer Additional FDs

- An FD F_{new} is *implied by* a set of FDs F if F_{new} holds whenever all FDs in F hold.

- F^+ = *closure of F* is the set of all FDs that are implied by F. (includes "trivial dependencies")

Rules of Inference

- Armstrong's Axioms (AA):
  - X, Y, Z are <u>sets</u> of attributes
    - *Reflexivity*: If $X \subseteq Y$, then Y -> X
    - *Augmentation*: If X -> Y, then XZ -> YZ for any Z
    - *Transitivity*: If X -> Y and Y -> Z, then X -> Z

- These are *sound* and *complete* inference rules for FDs!
  - i.e., using AA you can compute all the FDs in F^+ and only these FDs.

- Some additional rules (that follow from AA):
  - *Union*: If X -> Y and X -> Z, then X -> YZ
  - *Decomposition*: If X -> YZ, then X -> Y and X -> Z

Common Mistakes of Inferences

Is the following correct?

If (X, Y) -> Z,   then X -> Z and Y -> Z

X	Y	Z
1	2	1
2	2	2
1	1	2
1	2	1

WRONG!!!

Remember:

- Both Union and Decomposition rules ONLY union/decompose the right-hand-side of FDs.
- Never union/decompose the left-hand-side of FDs.

Example I

Inference rules:

- *Reflexivity*: If $X \subseteq Y$, then $Y \rightarrow X$
- *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Given:

–Relation R = {A,B,C,G,H,I}

–FDs: $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$

Questions: Prove the following FDs on R

(1) A->H

(2) $AG \rightarrow I$

(3) $CG \rightarrow HI$

Example II

- **Contracts(*<u>cid</u>,sid,jid,did,pid,qty,value*), and:**
  - C is the key: C -> CSJDPQV
  - Proj purchases each part using single contract: JP -> C
  - Dept purchases at most 1 part from a supplier: SD -> P
- **Question: Prove SDJ -> CSJDPQV**

Example II

- **Contracts(*<u>cid</u>,sid,jid,did,pid,qty,value*), and:**
  - C is the key: C -> CSJDPQV
  - Proj purchases each part using single contract: JP -> C
  - Dept purchases at most 1 part from a supplier: SD -> P
- **Question: Prove SDJ -> CSJDPQV**
  1. JP -> C, C -> CSJDPQV imply JP -> CSJDPQV (by transitivity)
  2. SD -> P implies SDJ -> JP (by augmentation)
  3. SDJ -> JP, JP -> CSJDPQV imply SDJ -> CSJDPQV (by transitivity). Thus SDJ is a superkey.

Q: can you now infer that SD -> **CSDPQV (i.e., drop J on both sides)?**

No! FD inference is not like arithmetic multiplication.

FDs are NOT just for solving redundancy problem

- Use FDs to infer new FDs
- Use FDs to determine keys

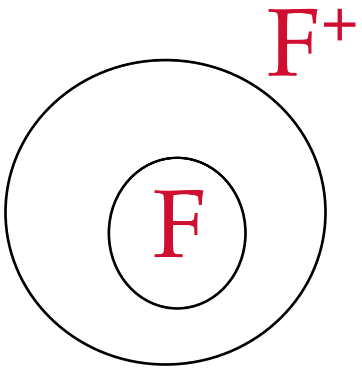
Use FDs to Determine Keys

- if “K->all attributes of R”, then K is a *superkey* for R
  - K is a superkey because it does not require K to be *minimal*.
- FDs are a generalization of keys.
  - FDs are NOT necessarily to be key constraints.

Recall: Closure of FDs

- **An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.**

- **F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")**



Computing FD/Attribute Closure

- Good news:
  - A new FD $X \to Y$ can be inferred from a set of FDs F $\Leftrightarrow$ Infer $X \to Y \in F^+$.
- Bad news:
  - Computing F^+ is inefficient (3 AA rules need to be applied repeatedly)

- An efficient check:
  - Compute *attribute closure* of X (denoted X^+) wrt *F*.
  - X^+ = a set of the attributes A such that X -> A is in F^+
  - How to compute X^+?
    - Initialize $X^+ := X$;
    - Repeat: if there is an FD U -> V in *F* such that U is in X^+, then add V to X^+.
    - Terminate the iterations when there is no change on X^+.
  - If $Y \in X^+$, then $X \to Y$ is in F^+ (i.e., X->Y can be inferred by F)

Attribute Closure (Example 1)

- **R(ABCDE)**
- **F={A->D, D->B, B->C, E->B}**
- **What's A$^+$, D$^+$, E$^+$, ACE$^+$?**

Attribute Closure (Example 2)

- **R = {A, B, C, D, E}**
- **F = { B -> CD, D -> E, B -> A, E -> C, AD -> B }**
- **Can B->E be inferred by F?**
  - I.e., Is B -> E in F^+?
  - I.e., Is E in B^+?

B^+ = B

B^+ = BCD

B^+ = BCDA

B^+ = BCDAE ... Yes!

FD Closure VS. Finding Key

- FD closure can be used to find the keys of a relation.
  - If X^+ = {all attributes of R} , then X is a superkey for R.
  - *Question:* How to check if the superkey X is a *candidate* key?
  - *Answer:* check whether X is minimal
  - This is equivalent to checking if any subset Y of X satisfies:

$$Y^+ = \{\text{all attributes of R}\}$$

    - If there is no such subset Y, then X is a candidate key (because X is minimal)
    - Otherwise, X is NOT a candidate key

Key Calculation

- **R = {A, B, C, D, E}**
- **F = { B -> CD, D -> E, B -> A, E -> C, AD -> B }**
  - Is D a superkey of R?
  - Is B a superkey of R?
  - Is B a candidate key of R?
  - Is AD a superkey of R?
  - Is AD a candidate key of R?
  - Is ADE a candidate key of R?

Search Space for Candidate Keys

- **The brute-force approach**
  - K attributes, $2^k - 1$ attribute sets to check
  - For example, given 3 attributes {A, B, C}, we have to check 7 candidates for the worst case
    - 1 attribute: {A}, {B}, {C}
    - 2 attributes: {A, B}, {B, C}, {A, C}
    - 3 attributes: {A, B, C}
    - We have to compute the FD closure of each candidate

Too slow!

How to Determine Candidate Keys?

- **An efficient solution:**
  - Group attributes into three categories
    - *L* category: attributes <u>only</u> appear at the <u>left</u> side of all given FDs
    - *R* category: attributes <u>only</u> appear at the <u>right</u> side of all given FDs
    - *M* category: attributes that appear at left side of some FDs, and right side of some other FDs.
  - The principle:
    - Attributes in L category: each candidate key should include <u>ALL</u> attribute in L category;
    - Attributes in M category may or may not be part of keys. Need computation of attribute closure to decide.
    - Attributes in R category should NOT be included in any key

Determine the keys (example 1)

- **Database : R(A, B, C, D)**
- **FDs: (AB -> C, C-> B, C-> D)**
- **What are the candidate keys of R?**

Determine the keys (example 1)

- **Database : R(A, B, C, D)**
- **FDs: (AB -> C, C-> B, C-> D)**
- **What are the candidate keys of R?**

L	**M**	**R**
A	B, C	D

  – A^+ = {A}: A cannot be a superkey (and thus a candidate key)
  – B^+ = {B}: B cannot be a superkey (and thus a candidate key)
  – C^+ = {C, B, D}: C cannot be a superkey (and thus a candidate key)
  – D^+ : no need to calculate as it is labeled with "R"
  – AB^+ ={A,B,C,D}
  – AC^+ ={A,C,B,D}
  – BC^+ ={B,C,D}
  – Keys: {A,B}, {A,C}

Determine the keys (example 2)

- **Database : R(A, B, C)**
- **FDs: (A->B, B-> C, C-> A)**
- **What are the candidate keys of R?**

L	M	R
	A, B, C	

  - $A^+ = \{A,B,C\}$
  - $B^+ = \{A,B,C\}$
  - $C^+ = \{A,B,C\}$
  - Keys: {A}, {B}, {C}