Schema Normalization

CS 564- Fall 2015

How To Build a DB Application

- Pick an application
- Figure out what to model (ER model)
  - Output: ER diagram
- Transform the ER diagram to a relational schema
- Refine the relational schema (normalization)
- Now ready to implement the schema and load the data!

Motivating Example

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

- What is the primary key?
  - (SSN, PhoneNumber)
- What is the problem with this schema?

Motivating Example

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

Problems:

- redundant storage
- update: change the age of Paris?
- insert: what if a person has no phone number?
- delete: what if Arun deletes his phone number?

Solution: Decomposition

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

General Solution

- identify “bad” schemas
  - functional dependencies (**FDs**)
- decompose the tables (normalize) using the FDs specified
- decomposition should be used judiciously:
  - normal forms (**BCNF**, **3NF**) guarantee against some forms of redundancy
  - does decomposition cause any problems?
    - Lossless join
    - Dependency preservation

FUNCTIONAL DEPENDENCIES

FD: DEFINITION

- FDs are a form of constraint
- generalize the concept of keys

If two tuples agree on the attributes

$$A_1, A_2, \dots, An$$

then they must agree on the attributes

$$B_1, B_2, \dots, B_m$$

Formally:

$$A_1, A_2, \dots, A_n \longrightarrow B_1, B_2, \dots, B_m$$

FD: EXAMPLE 1

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

- $SSN \longrightarrow name, age$
- $SSN, age \longrightarrow name$
- $SSN \nrightarrow phoneNumber$

FD: EXAMPLE 2

studentID	semester	courseNo	section	instructor
124434	4	CS 564	1	Paris
546364	4	CS 564	2	Arun
999492	6	CS 764	1	Anhai
183349	6	CS 784	1	Jeff

- $courseNo, section \longrightarrow instructor$
- $studentID \longrightarrow semester$

How Do We Infer FDs?

- What FDs are valid for a relational schema?
  - think from an application point of view
- An FD is
  - an inherent property of an application
  - not something we can infer from a set of tuples
- Given a table with a set of tuples
  - we can confirm that a FD **seems** to be valid
  - to infer that a FD is **definitely** invalid
  - we can **never** prove that a FD is valid

EXAMPLE 3

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supplies	59

To confirm whether $name \longrightarrow department$

- erase all other columns
- check that the relationship name-department is many-one!

WHY FDS?

- keys are special cases of FDs
- more integrity constraints for the application
- having FDs will help us detect that a table is "bad", and how to decompose the table

More on FDs

- If the following FDs hold:
  - $A \longrightarrow B$
  - $B \longrightarrow C$
- then the following FD is also true:
  - $A \longrightarrow C$
- This means that there are more FDs that can be found! How?
  - Armstrong's Axioms

Armstrong's Axioms: 1

Reflexivity

For any subset $X \subseteq \{A_1, .., A_n\}$:

$$A_1, A_2, \dots, A_n \longrightarrow X$$

- Examples
  - $A, B \longrightarrow B$
  - $A, B, C \longrightarrow A, B$
  - $A, B, C \longrightarrow A, B, C$

Armstrong's Axioms: 2

Augmentation

For any attribute sets X, Y, Z :

if $X \longrightarrow Y$ then $X, Z \longrightarrow Y, Z$

- Examples
  - $A \longrightarrow B$ implies $A, C \longrightarrow B, C$
  - $A, B \longrightarrow C$ implies $A, B, C \longrightarrow C$

Armstrong's Axioms: 3

Transitivity

For any attribute sets X, Y, Z :

if $X \longrightarrow Y$ and $Y \longrightarrow Z$ then $X \longrightarrow Z$

- Examples
  - $A \longrightarrow B$ and $B \longrightarrow C$ imply $A \longrightarrow C$
  - $A \longrightarrow C, D$ and $C, D \longrightarrow E$ imply $A \longrightarrow E$

APPLYING ARMSTRONG'S AXIOMS

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

- $name \longrightarrow color$
- $category \longrightarrow department$
- $color, category \longrightarrow price$

Inferred FDs:

- $name, category \longrightarrow price$
  - (1) Augmentation, (2) Transitivity
- $name, category \longrightarrow color$
  - (1) Reflexivity, (2) Transitivity

APPLYING ARMSTRONG'S AXIOMS

FD Closure

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

Armstrong's axioms are:

- **sound**: any FD generated by an axiom belongs in F^+
- **complete**: repeated application of the axioms will generate all FDs in F^+

Closure of Attribute Sets

Attribute Closure

If X is an attribute set, the closure X^+ is the set of all attributes B such that:

$$X \longrightarrow B$$

In other words, X^+ includes all attributes that are functionally determined from X

EXAMPLE

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

- $name \longrightarrow color$
- $category \longrightarrow department$
- $color, category \longrightarrow price$

Attribute Closure:

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

WHY IS CLOSURE NEEDED?

- Does $X \longrightarrow Y$ hold?
  - check if Y $\subseteq X^+$
- Compute the closure F^+ of FDs
  - for each subset of attributes X, compute X^+
  - for each subset of attributes Y $\subseteq X^+$, output the FD $X \longrightarrow Y$

Closure Algorithm

Let $X = \{A_1, A_2, \dots, A_n\}$

Until X doesn't change **repeat**:

if $B_1, B_2, \dots, B_m \longrightarrow C$ is a FD **and**

$B_1, B_2, \dots, B_m$ are all in X

then add C to X

EXAMPLE

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

- $A, B \longrightarrow C$
- $A, D \longrightarrow E$
- $B \longrightarrow D$
- $A, F \longrightarrow B$

Compute:

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

KEYS & SUPERKEYS

Relation **R**

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

$$A_1, A_2, \ldots, A_n \longrightarrow B$$

- **key:** a minimal superkey
  - none of its subsets functionally determines all attributes of **R**

Computing Keys & Superkeys

- Compute X^+ for all sets of attributes X
- If $X^+ = all\ attributes$, then X is a superkey
- If no subset of X is a superkey, then X is also a key

EXAMPLE

Product(name, category, price, color)

- $name \longrightarrow color$
- $color, category \longrightarrow price$

Superkeys:

- $\{name, category\}, \{name, category, price\}$
  $\{name, category, color\}, \{name, category, price, color\}$

Keys:

- $\{name, category\}$

MANY KEYS?

Q: Is it possible to have many keys in a relation **R** ?

YES!! Take relation **R**(A, B, C)with FDs

- $A, B \longrightarrow C$
- $A, C \longrightarrow B$

RECAP

- FDs and (super)keys
- Reasoning with FDs:
  - given a set of FDs, infer all implied FDs
  - given a set of attributes X, infer all attributes that are functionally determined by X
- Next we will look at how to use them to detect that a table is "bad"

BCNF Decomposition

BOYCE-CODD NORMAL FORM (BCNF)

A relation **R** is in **BCNF** if whenever $X \longrightarrow B$ is a non-trivial FD, then X is a superkey in **R**

Equivalent definition: for every attribute set X

- either $X^+ = X$
- or $X^+ = all\ attributes$

BCNF Example 1

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

$$SSN \longrightarrow name, age$$

- **key** = $\{SSN, phoneNumber\}$
- $SSN \longrightarrow name, age$ is a "bad" dependency
- The above relation is **not** in BCNF!

BCNF Example 2

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

$$SSN \longrightarrow name, age$$

- **key** = $\{SSN\}$
- The above relation is in BCNF!

BCNF Example 3

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

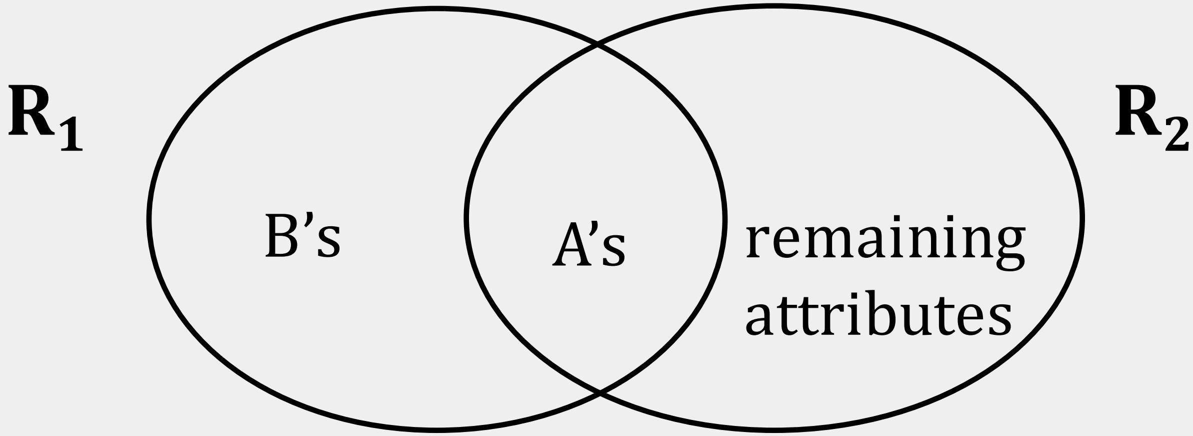
- **key** = $\{SSN, phoneNumber\}$
- The above relation is in BCNF!
- **Q**: can we have a binary relation that is not in BCNF?

BCNF Decomposition

- Find an FD that violates the BCNF condition

$$A_1, A_2, \dots, A_n \longrightarrow B_1, B_2, \dots, B_m$$

- Decompose **R** to $\mathbf{R}_1$ and $\mathbf{R}_2$:

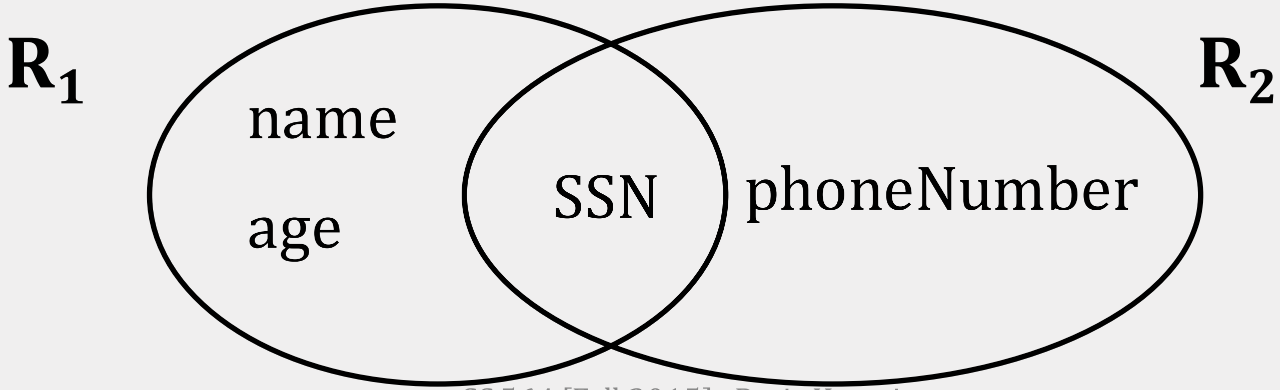


- Continue until no BCNF violations are left

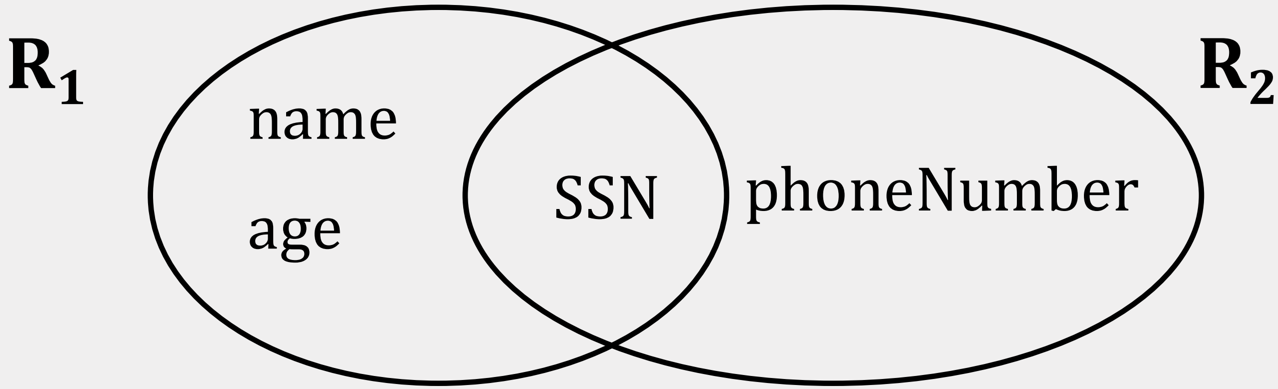
DECOMPOSITION EXAMPLE

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

- The FD $SSN \longrightarrow name, age$ violates BCNF
- Split into two relations $\mathbf{R_1}, \mathbf{R_2}$ as follows:



DECOMPOSITION EXAMPLE



$$SSN \longrightarrow name, age$$

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

BCNF EXAMPLE

Person(SSN, name, age, canDrink, phoneNumber)

- $SSN \longrightarrow name, age$
- $age \longrightarrow canDrink$

DECOMPOSITION PROPERTIES

DECOMPOSITION IN GENERAL

Let **R**(A_1, …, A_n). To decompose, create:

- $\mathbf{R_1}$(B_1, .., B_m)
- $\mathbf{R_2}$(C_1,…, C_l)
- where $\{B_1, \dots, B_m\} \cup \{C_1, \dots, C_l\} = \{A_1, \dots A_n\}$

Then:

- $\mathbf{R_1}$ is the projection of **R** onto B_1, .., B_m
- $\mathbf{R_2}$ is the projection of **R** onto C_1, .., C_l

PROPERTIES

1. minimize redundancy
2. avoid information loss
3. preserve functional dependencies
4. ensure good query performance

Information Loss Example

name	age	phoneNumber
Paris	24	608-374-8422
John	24	608-321-1163
Arun	20	206-473-8221

Decompose into:

- $\mathbf{R_1}$(name, age)
- $\mathbf{R_2}$(age, phoneNumber)

name	age
Paris	24
John	24
Arun	20

age	phoneNumber
24	608-374-8422
24	608-321-1163
20	206-473-8221

Can we put it back together?

Lossless-Join Decomposition

$$
\begin{tikzcd}
 & \mathbf{R}(A, B, C) \arrow[dl] \arrow[dr] & & \text{decompose} \\
\mathbf{R_1}(A, B) \arrow[dr] & & \mathbf{R_2}(B, C) \arrow[dl] & \\
 & \mathbf{R'}(A, B, C) & & \text{recover}
\end{tikzcd}
$$

The decomposition is lossless-join if
R' is the same as **R**

FD PRESERVING

- Given a relation $\mathbf{R}$ and a set of FDs F, decompose $\mathbf{R}$ into $\mathbf{R_1}$ and $\mathbf{R_2}$
- Suppose
  - $\mathbf{R_1}$ has a set of FDs F_1
  - $\mathbf{R_2}$ has a set of FDs F_2
  - F_1 and F_2 are computed from F

The decomposition is dependency preserving if by enforcing F_1 over $\mathbf{R_1}$ and F_2 over $\mathbf{R_2}$, we can enforce F over $\mathbf{R}$

GOOD EXAMPLE

Person(SSN, name, age, canDrink)

- $SSN \longrightarrow name, age$
- $age \longrightarrow canDrink$

Decomposes into:

- $\mathbf{R_1}$(SSN, name, age)
  - $SSN \longrightarrow name, age$
- $\mathbf{R_2}$(age, canDrink)
  - $age \longrightarrow canDrink$

Bad Example

R(A, B, C)

- $A \longrightarrow B$
- $B, C \longrightarrow A$

Decomposes into:

- $\mathbf{R_1}$(A, B)
  - $A \longrightarrow B$
- $\mathbf{R_2}$(A, C)
  - no FDs here!!

$\mathbf{R_1}$

A	B
a_1	b
a_2	b

$\mathbf{R_2}$

A	C
a_1	c
a_2	c

recover

A	B	C
a_1	b	c
a_2	b	c

The recovered table violates $B, C \longrightarrow A$

DECOMPOSITION

- When decomposing a relation **R**, we want to achieve good properties
- These properties can be conflicting
- BCNF decomposition achieves some of these:
  - removes certain types of redundancy
  - is **lossless-join**
  - is **not always** dependency preserving

WHY IS BCNF LOSSLESS-JOIN?

Example:

R(A, B, C) with $A \longrightarrow B$ decomposes into:

$\mathbf{R_1}$(A, B) and $\mathbf{R_2}$(A, C)

- Suppose tuple (a,b,c) is in the recovered **R'**
- Then, (a,b) in $\mathbf{R_1}$ and (a,c) in $\mathbf{R_2}$
- But then (a,b',c) is in **R**
- Since $A \longrightarrow B$ it must be that b' = b
- So (a,b,c) is also in **R**!

WHY IS BCNF NOT FD PRESERVING?

R(A, B, C)

- $A \longrightarrow B$
- $B, C \longrightarrow A$

The BCNF decomposition is:

- $\mathbf{R_1}$(A, B) with FD $A \longrightarrow B$
- $\mathbf{R_2}$(A, C) with no FDs

There may not exist any BCNF decomposition that is FD preserving!

Normal Forms

BCNF is what we call a normal form

Other normal forms exist:

- 1NF: flat tables (atomic values)
- 2NF
- 3NF
- BCNF
- 4NF
- …

more restrictive

THIRD NORMAL FORM (3NF)

3NF DEFINITION

A relation **R** is in **3NF** if whenever $X \longrightarrow A$, one of the following is true:

- $A \in X$ (trivial FD)
- X is a superkey
- A is part of some key of **R** (prime attribute)

BCNF implies 3NF

3NF CONTINUED

- Example: **R**(A, B, C) with $A, B \longrightarrow C$ and $C \longrightarrow A$
  - is in 3NF. Why?
  - is not in BCNF. Why?

- Compromise used when BCNF not achievable: *aim for BCNF and settle for 3NF*
- Lossless-join and dependency-preserving decomposition of **R** into a collection of 3NF relations is always possible!

3NF DECOMPOSITION

- The algorithm for BCNF decomposition can be used to get a lossless-join decomposition into 3NF
- We can typically stop earlier!
- To ensure dependency preservation as well, instead of the given set of FDs F, we use a minimal (or canonical) cover for F

Minimal Cover For FDs

- minimal cover G for a set of FDs F:
  - $G^+ = F^+$
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes
  - The LHS of each FD is unique

- The minimal cover gives a lossless-join and dependency-preserving decomposition algorithm!

EXAMPLE

Example:

- $A \longrightarrow B$
- $A, B, C, D \longrightarrow E$
- $E, F \longrightarrow G, H$
- $A, C, D, F \longrightarrow E, G$

The minimal cover is the following:

- $A \longrightarrow B$
- $A, C, D \longrightarrow E$
- $E, F \longrightarrow G, H$

3NF Algorithm

1. Find a lossless-join 3NF decomposition (that might violate some FDs)
2. Compute a minimal cover F
3. Find the FDs in F that are not preserved
4. For each non-preserved FD $X \longrightarrow A$ add a new relation R(X, A)

Is Normalization Always Good?

- Example: suppose A and B are always used together, but normalization says they should be in different tables
  - decomposition might produce unacceptable performance loss
- Example: data warehouses
  - huge historical DBs, rarely updated after creation
  - joins expensive or impractical
- Everyday DBs: aim for BCNF, settle for 3NF!

RECAP

- Bad schemas lead to redundancy
  - redundant storage, update, insert, and delete anomaly
- To "correct" bad schemas: decompose relations
  - must be a lossless-join decomposition
  - would like dependency-preserving decompositions
- Desired normal forms
  - **BCNF**: only superkey FDs
  - **3NF**: superkey FDs + dependencies with prime attributes on the RHS