Better Designs Exist

Break the relation into two:

	SSN	Address
	123-321-99	10 Addr
Alex	123-321-99	10 Green
Sally	909-438-44	431 Purple

Identify

?

Address

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

A1
A1
A2
A2

Reminder

Tutorial #2.

→ Today! 4:30-5:30pm

1302 SC.

P-Fun Topics?

- Translation ER to Rel. model.
- RAT 10%

R.A. → SQL → SQLite ⇒ R.A. → SQLite

- Attr closure (today) - BCNF (tod.)
- F.D. closuer (") - " " "

Functional Dependencies

- A form of constraint (hence, part of the schema)
- Finding them is part of the database design
- Used heavily in schema refinement

(SSN) Name | Addr | Phone

10 green

10 green

Name | Address

Definition:

If two tuples agree on the attributes

T1 Alex ...

T2 Alex ...

Name

$A_1, A_2, \ldots A_n$

then they must also agree on the attributes

Address

$B_1, B_2, \ldots B_m$

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

- EmpID $\longrightarrow$ Name, Phone, Position
- Position $\longrightarrow$ Phone
- but Phone $\not\longrightarrow$ Position

1234

clerk, Lawyer

In General

- To check if A $\longrightarrow$ B violation:

Erase all other columns

phone ← position

…	A	…	B	
	X1		Y1	
	X2		Y2	
	…		…	

one ← many / one

- check if the remaining relation is many-one (called ***functional*** in mathematics)

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	lawyer

No violation for Pos → Phone

More examples:

Product: name → price, manufacturer

Person: ssn → name, age

Company: name → stock price, president

Q: From this, can you conclude phone → SSN?

a phone is only used by ONE person.

SSN		Phone Number
123-321-99	Alex	(201) 555-1234
123-321-99	Alex	(206) 572-4312
909-438-44		(908) 464-0028
909-438-44		(212) 555-4000
123-321-88	Alex Junior	(201) 555-1234

F.D. stated at schema design

⇒ assertion

Relation Keys

NetID	Name	addr	dept	age	...

- After defining FDs, we can now define keys
- Key of a relation R is a set of attributes that
  - functionally determines all attributes of R
  - none of its subsets determines all attributes of R
- Superkey
  - a set of attributes that contains a key
- We will need to know the keys of the relations in a DB schema, so that we can refine the schema

{N, A} → {NetID, dept, age, } ...

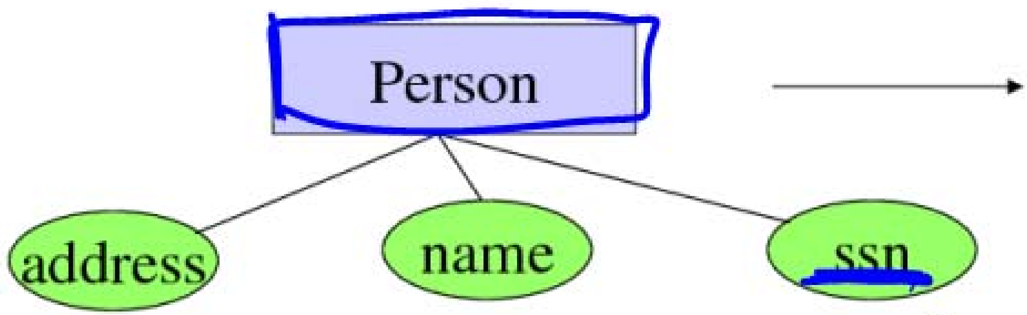
	(Name, Addr)	(Netid)	(NetID, dept)
key	✓	✓	✗
s.key	✓	✓	✓

Finding the Keys of a Relation

Given a relation constructed from an E/R diagram, what is its key?

Rules:

1. If the relation comes from an entity set,
   the key of the relation is the set of attributes which is the key of the entity set.

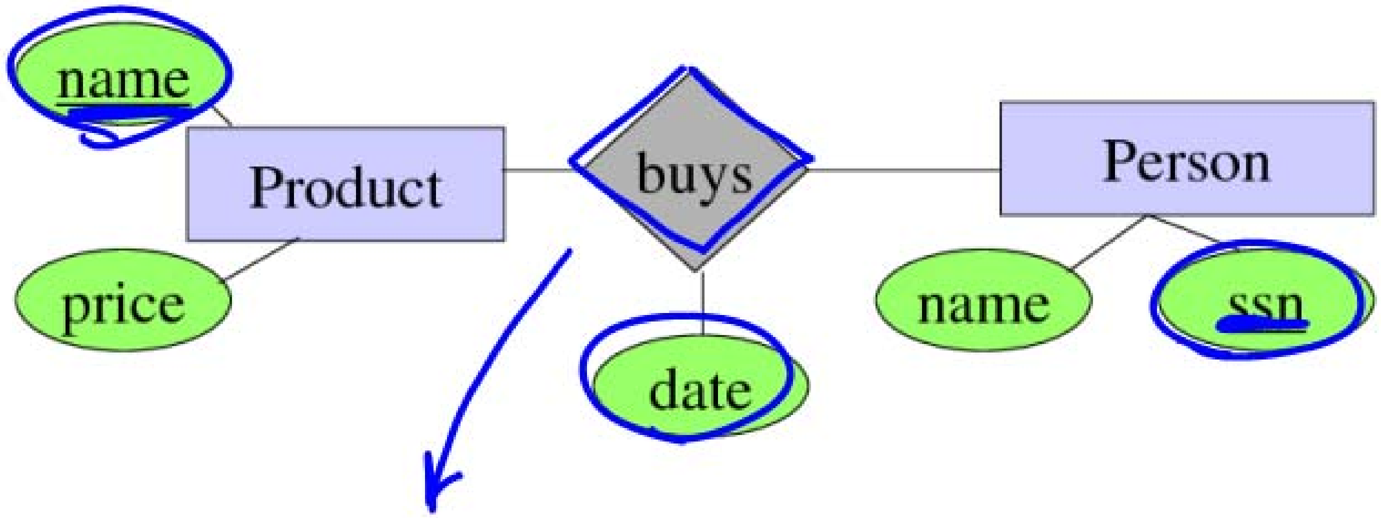


Person(address, name, ssn)

Finding the Keys

Rules:

2. If the relation comes from a many-many relationship, the key of the relation include the set of all attribute keys in the relations corresponding to the entity sets (and additional attributes if necessary)

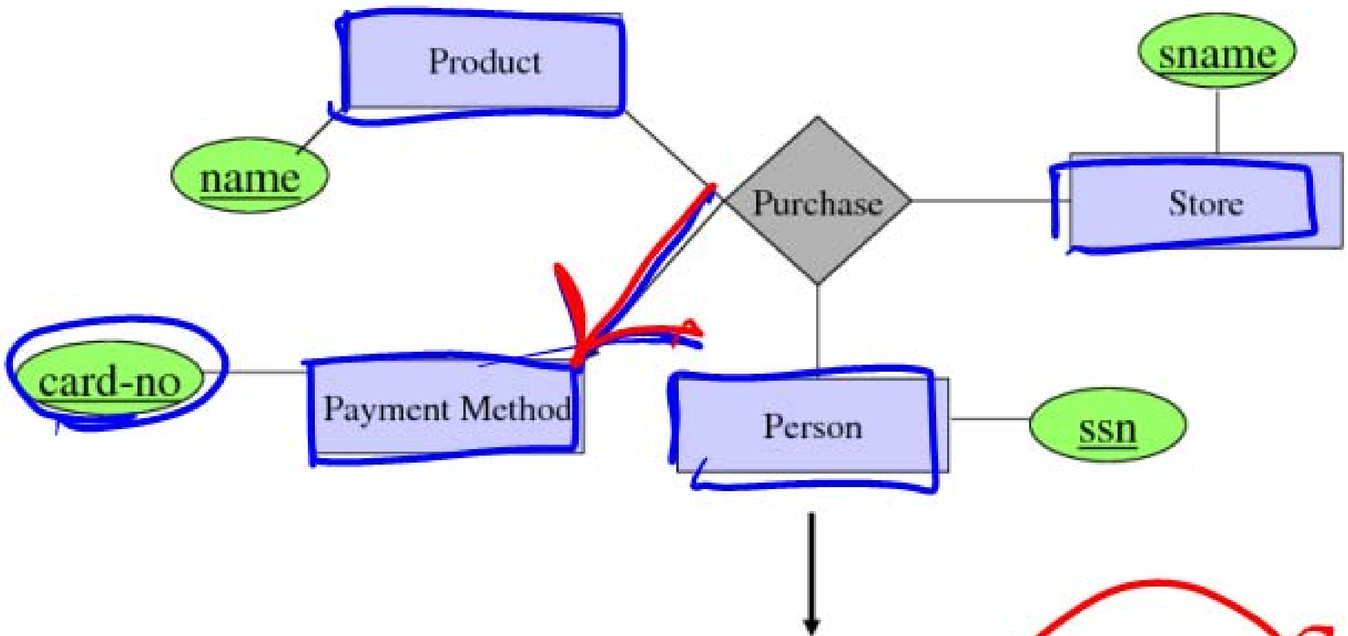


buys(<u>name, ssn</u>, date)

prod person

Finding the Keys

But: if there is an arrow from the relationship to E, then
we don't need the key of E as part of the relation key.



Purchase(name , sname, ssn, card-no)

Finding the Keys

More specific rules:

- Many-one, one-many, one-one relationships
- Multi-way relationships
- Weak entity sets

(Try to find them yourself)

Reasoning with FDs

1) closure of FD sets
2) closure of attribute sets

Closure of FD sets

You

who gives FDs ?

- Given a relation schema R & a set S of FDs
  - is the FD f logically implied by S?
- Example
  - R = {A,B,C,G,H,I} 6 attrs.
  - S = A → B, A → C, CG → H, CG → I, B → H
  - would A → H be logically implied? new rule
  - yes (you can prove this, using the definition of FD)

Name → age
age → drikable

Name
↓
drinkable

- Closure of S: S+ = all FDs logically implied by S

such as

A → H.

- How to compute S+?
  - we can use Armstrong's axioms

Armstrong's Axioms Rules

proven by def.

- Reflexivity rule NetID → dept. addr
  - A1A2...An ➔ a subset of A1A2...An ⇒ NetID → dept
- Augmentation rule Name.NetID → dept. addr. Name
  - A1A2...An ➔ B1B2...Bm, then
    A1A2...An C1C2..Ck ➔ B1B2...Bm C1C2...Ck
- Transitivity rule
  - A1A2...An ➔ B1B2...Bm and
    B1B2...Bm ➔ C1C2...Ck, then
    A1A2...An ➔ C1C2...Ck

Name → age
age → drink
⇒ Name → drink

Inferring S+ using Armstrong's Axioms

$S = \{ A \xrightarrow{f1} B,\ B \xrightarrow{f2} C,\ AC \xrightarrow{f3} D \}$

S^+ ?

- S+ = S
- Loop
  - foreach f in S, apply reflexivity and augment. rules
  - add the new FDs to S+
  - foreach pair of FDs in S, apply the transitivity rule
  - add the new FD to S+
- Until S+ does not change any further

"form changing"

so that $B \Rightarrow B$

$f_1 . f_2$. Tr $\Rightarrow + A \to C$.

(want to use $AC \to D$)

$f2: AB \to AC$

$f1: AA \to AB$

$\Rightarrow AA \to D \Rightarrow A \to D$

Name ______________ NetID ______________

Q1: What do you like best of this cls. that we must keep?

Q2: What ... dislike .... go?

Additional Rules

- Union rule (aug)
  - $X \rightarrow Y$ and $X \rightarrow Z$, then $\boxed{X \rightarrow YZ}$

    netid dept netid addr netid dept addr
  - (X, Y, Z are sets of attributes)

  $XX \rightarrow XY$
  $XY \rightarrow YZ$ $\Rightarrow XX \rightarrow YZ$

- Decomposition rule
  - $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Pseudo-transitivity rule
  - $X \rightarrow Y$ and $YZ \rightarrow U$, then $XZ \rightarrow U$
- These rules can be inferred from Armstrong's axioms

Closure of a Set of Attributes

(name, addr) → ?

Given a set of attributes {*A1, …, An*} and a set of dependencies S.
(You)
Problem: find all attributes *B* such that:

any relation which satisfies S also satisfies:

A1, …, An → *B*

The **closure** of *{A1, …, An}*, denoted *{A1, …, An}*$^+$,
is the set of all such attributes *B*

?

We will discuss the motivations for attribute closures soon

Is {name, addr} a key?

$\boxed{\{name, addr\}^+}$ ⇒ all attr.

Algorithm to Compute Closure

Start with X={A1, …, An}. {name, addr}

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

if $B_1, B_2, \ldots B_n \longrightarrow C$ is in S, **and**

$B_1, B_2, \ldots B_n$ are all in X, **and**

C is not in X

then

add C to X. $X = X + \{C\}$

Example

$R: \langle A, B, C, D, E, F \rangle$ Is (A, F) a key?

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

Name Addr $\rightarrow$ age

✓ $(A,F)^+ = \{A, \ldots, F\}$

✗ $(A)^+ = \{\ldots\}$

✗ $(F)^+ = \{\ldots\}$

Closure of {A,B}: X = {A, B, C, D, E}

Closure of {A, F}: X = {A, F, B, D, C, E}

	(A, F) (A) ?
$AF \rightarrow B$	A, B, F
$B \rightarrow D$	A, B, D, F
$AD \rightarrow E$	A B D E F
$AB \rightarrow C$	A B C D E F @stop

Usage for Attribute Closure

- Test if X is a superkey
  - compute X+, and check if X+ contains all attrs of R

- Check if $\underbrace{X \rightarrow Y}$ holds
  - by checking if Y is contained in X+

$$Y \subseteq X^+ \iff X \rightarrow Y$$

ER

$R \xrightarrow{?} R^*$

original

Normal Form

Desirable Properties of Schema Refinement

1) minimize redundancy
2) avoid info loss
3) preserve dependency
4) ensure good query performance

X

Normal Forms

x set. string.

x array. Float.

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

SQL. Ted Codd.

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...

Boyce-Codd Normal Form

BCNF ? NO

ssn	addr	phone
Alex	10 G	123
Alex	10 G	456

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

✓ ssn → addr Bad

Whenever there is a nontrivial FD $A_1, A_2, \ldots A_n \longrightarrow B$

for R, it is the case that $\{ A_1, A_2, \ldots A_n \}$ is a super-key for R.

then

ssn is superkey

In English (though a bit vague):

X ? ssn → ~~all attr~~ ssn, addr, phone

Whenever a set of attributes of *R* is determining another attribute, it should determine ***all*** attributes of *R*.

In Contrast

✓ ssn → addr

✓ ssn a key

yes

BCNF ?

ssn	addr
Alex	10 G
Alex	10 G

ssn	phone
Alex	123
Alex	456

Example

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN → Name

What are the keys?

Is it in BCNF?

Decompose it into BCNF

not BCNF

(addr)

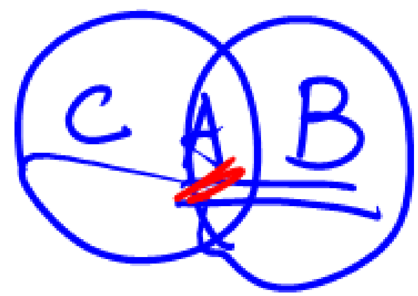
SSN	Name	Phone
A	B	C

$\underline{A \; B}$

T1

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN ⟶ Name



T2

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

$\underline{A \; C}$

What About This?

Name	Price	Category
Gizmo	$19.99	gadgets
OneClick	$24.99	camera

Name $\longrightarrow$ Price, Category