Relational Design Theory

CS411 Database Systems

Kevin C. Chang

Why Do We Learn This?

Motivation

- We have designed ER diagram, and translated it into a relational db schema R = set of R1, R2, ...
- Now what?
- We can do the following
  - specify all relevant constraints over R
  - implement R in SQL
  - start using it, making sure the constraints always remain valid
- However, R may not be well-designed, thus causing us a lot of problems

This a good design?

Persons with several phones:

(notid)

Address	SSN	Phone
10 Green	123-321-99	(201) 555-1234
10 Green	123-321-99	(206) 572-4312
431 Purple	909-438-44	(908) 464-0028
431 Purple	909-438-44	(212) 555-4000

Potential Problems

- Redundancy
- Update anomalies
- Deletion anomalies

How do We Obtain a Good Design?

- Start with the original db schema R (name, netid, phone)
- Transform it until we get a good design R*

good normal form

- Desirable properties for R*
  - must preserve the information of R
  - must have minimal amount of redundancy
  - must be [illegible] - if R is associated with a set of constraints C, then it should be easy to also check C over R*
  - (must also give good query performance)

OK, But ...

- How do we recognize a good design R*?
- How do we transform R into R*?
- What we need is the “theory” of …

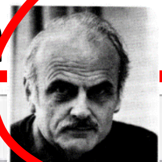
Normal Forms good design.

- DB gurus have developed many normal forms
- Most [illegible] ones 1. 2.
    - Boyce-Codd, 3rd, and 4th normal forms 5th?
- If R* is in one of these forms, then R* is guaranteed to achieve certain good properties
    - e.g., if R* is in Boyce-Codd NF, it is guaranteed to not have certain types of redundancy
- DB gurus have also developed algorithms to transform R into R* that is in some of these normal forms

Normal Forms (cont.)

- DB gurus have also discussed trade-offs among normal forms
- Thus, all we have to do is
  - learn these forms
  - transform R into R* in one of these forms
  - carefully evaluate the trade-offs
- Many of these normal forms are defined based on various constraints
  - functional dependencies and keys

Behind the Scene: Know whom we should blam



Normal form	Defined by	Brief definition
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)[12] 	Table faithfully represents a relation and has no "repeating groups"
Second normal form (2NF)	E.F. Codd (1971)[13]	No non-prime attribute in the table is functionally dependent on a part (proper subset) of a candidate key
Third normal form (3NF)	E.F. Codd (1971)[14]; see also Carlo Zaniolo's equivalent but differently-expressed definition (1982)[15]	Every non-prime attribute is non-transitively dependent on every key of the table
Boyce-Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)[16]	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977)[17] 	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)[18]	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)[19]	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	Chris Date, Hugh Darwen, and Nikos Lorentzos (2002)[20]	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

No ambiguous

Functional Dependencies

How values associate

Better Designs Exist

Address (NetID) SSN Phone

Break the relation into two:

R1

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

R2

SSN	Phone
123-321-99	(201) 555-2334
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

Functional Dependencies

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

More example

⊕ A_1 prod_title, A_2 store

"wii" "Am.com"

→ B_1 price, shipping,

Definition:

⊖ X

SSN → phone ?

prod_title → price, ship ?

If two tuples agree on the attributes

$A_1\ A_2 \ldots\ A_n$ {SSN}

then they must also agree on the attributes

$B_1\ B_2 \ldots B_n$ {Name}

Formally:

$$A_1\ A_2 \ldots\ A_n \rightarrow B_1\ B_2 \ldots B_n$$

SSN → Name

Examples

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

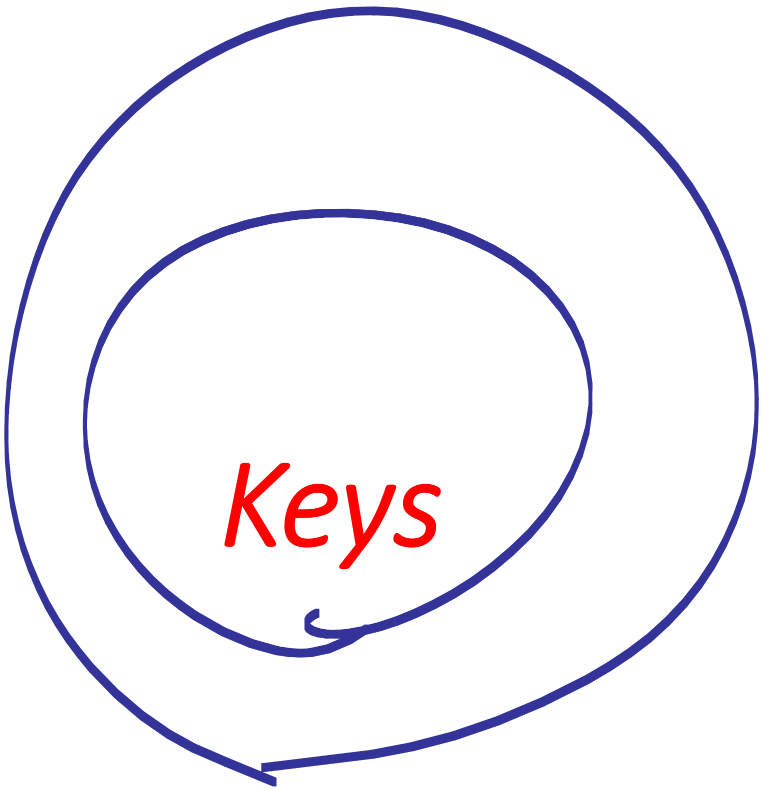
- EmpID → Name, Phone, Position ?
- Position → Phone ?
- Phone → Position ?

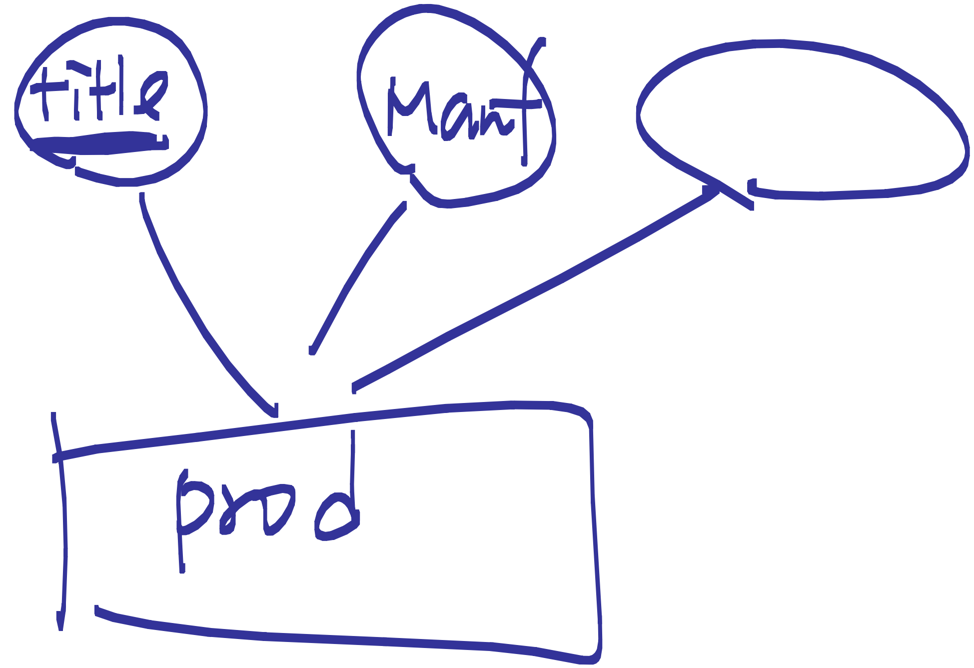
In General

- To check if $A \rightarrow B$ violation:
- Ignore all other columns

…	A	…	B	
	X1		Y1	
	X2		Y2	
	…		…	

- Check if the remaining relation is many-one– i.e., ***functional.***


Keys


title
Manf
prod

② Why a key unique?

Relation Keys

- 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 subset 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

{prod_title, store}

price, shipping, buyer,

superset of key

{title, store, price}

① Does a key always exist? Yes

worst → How all all attr if nothing smaller!

make up → "NetID"?

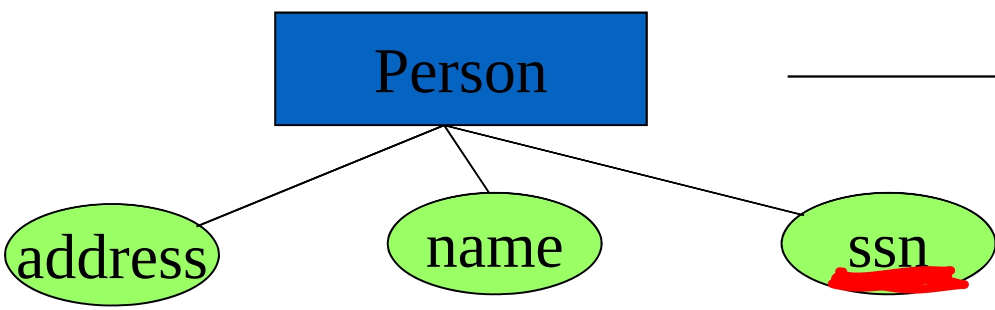
Finding the Keys of a Relation

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.

Ent. Set



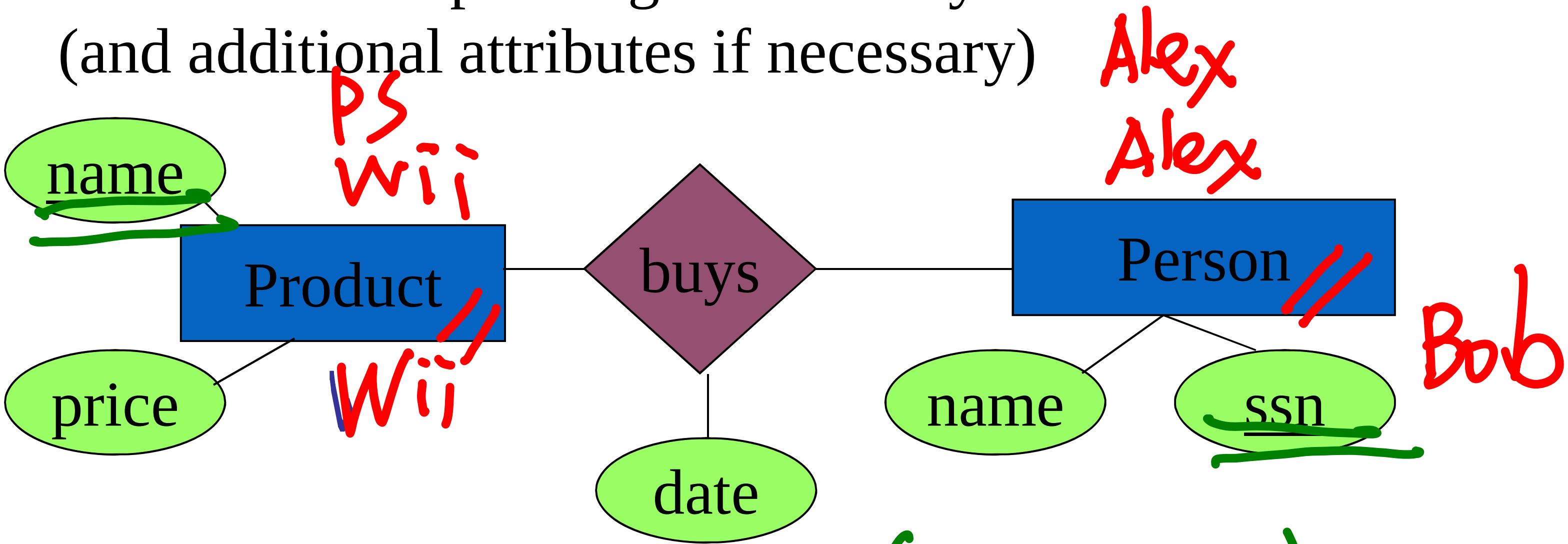
→ **Person(**address, name, <u>ssn</u>**)**

Finding the Keys

Many Many

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)

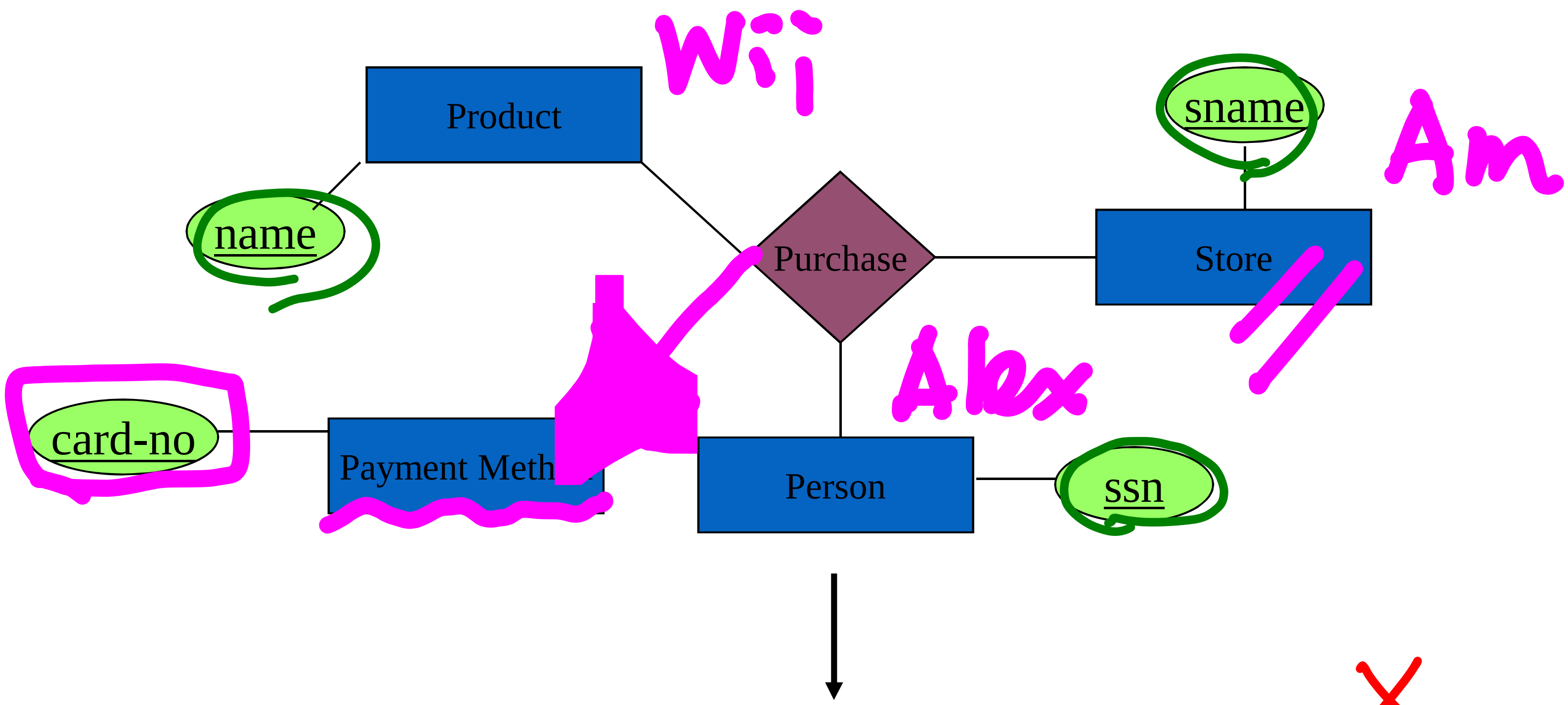


$\{name, ssn\} \rightarrow date,$

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

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.



X

Purchase(<u>name , sname, ssn</u>, card-no)

"one"

many

Finding the Keys

More specific rules:

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

(Try to find them yourself)

"Good Design"

Normal Forms

BCNF

Desirable Properties of Schema Refinement

- Minimize redundancy
- Avoid info loss
- Preserve dependency
- Ensure good query performance

Boyce, Codd. → No Good!

Boyce-Codd Normal Form

[name, SSN, phone] Criteria

- A relation R is in BCNF

R good?

Whenever there is a nontrivial FD for R ,

$$A_1 \; A_2 \; \ldots \; A_n \rightarrow B$$

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

① SSN → name

- That is:

② SSN → all attr ? NO SSN ↛ ph.

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

∵ ② : Bob Bob256 → 123-4567

Bad Bob Bob256 → 456-7890

Example

Name	City	LikeBeer
Alex	Champaign IL	Bud
Alex	Champaign IL	Michelob
Bob	Urbana IL	Bud
Bob	Urbana IL	Sam Adam

What are the dependencies? Name $\rightarrow$ City
What are the keys?
Is it in BCNF?