CSE 544
Principles of Database Management Systems

Fall 2016

Lecture 3 –Schema Normalization

Projects

- We have 27 teams
- Impossible to discuss projects at office hours tomorrow
- Instead, sign up on doodle for a 10’ slot on Monday.

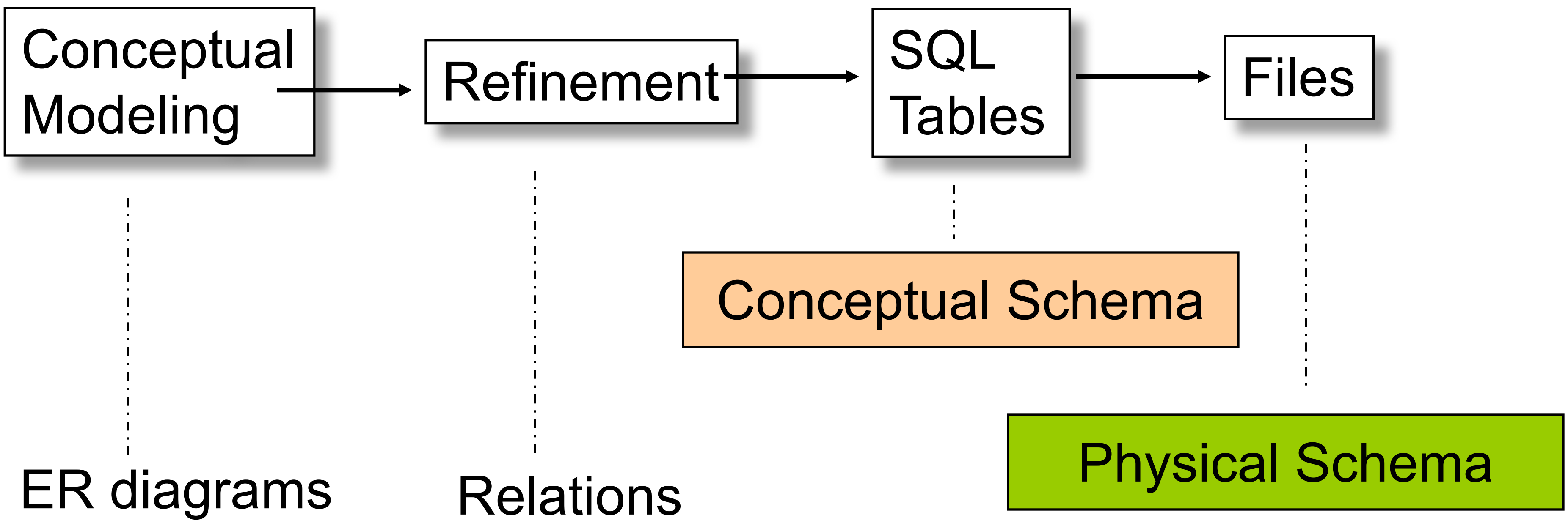
Database Design

- The relational model is great, but how do I design my database schema?

Outline

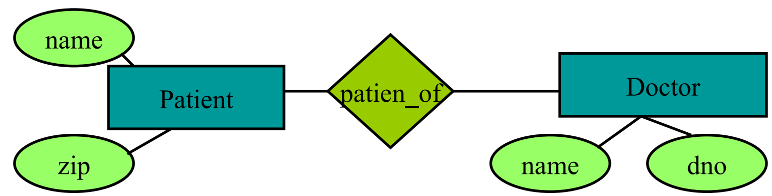
- Conceptual db design: entity-relationship model
- Problematic database designs
- Functional dependencies
- Normal forms and schema normalization

Database Design Process

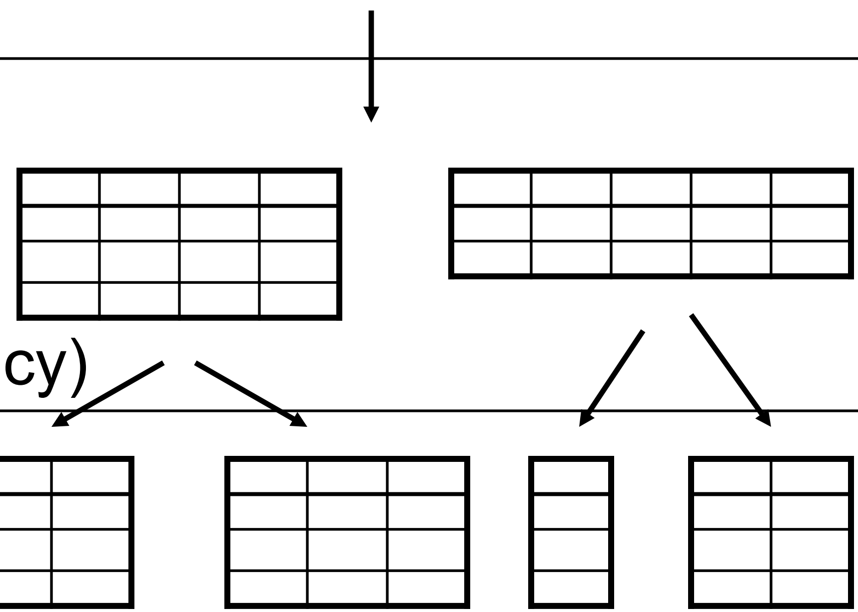


Conceptual Schema Design

Conceptual Model:

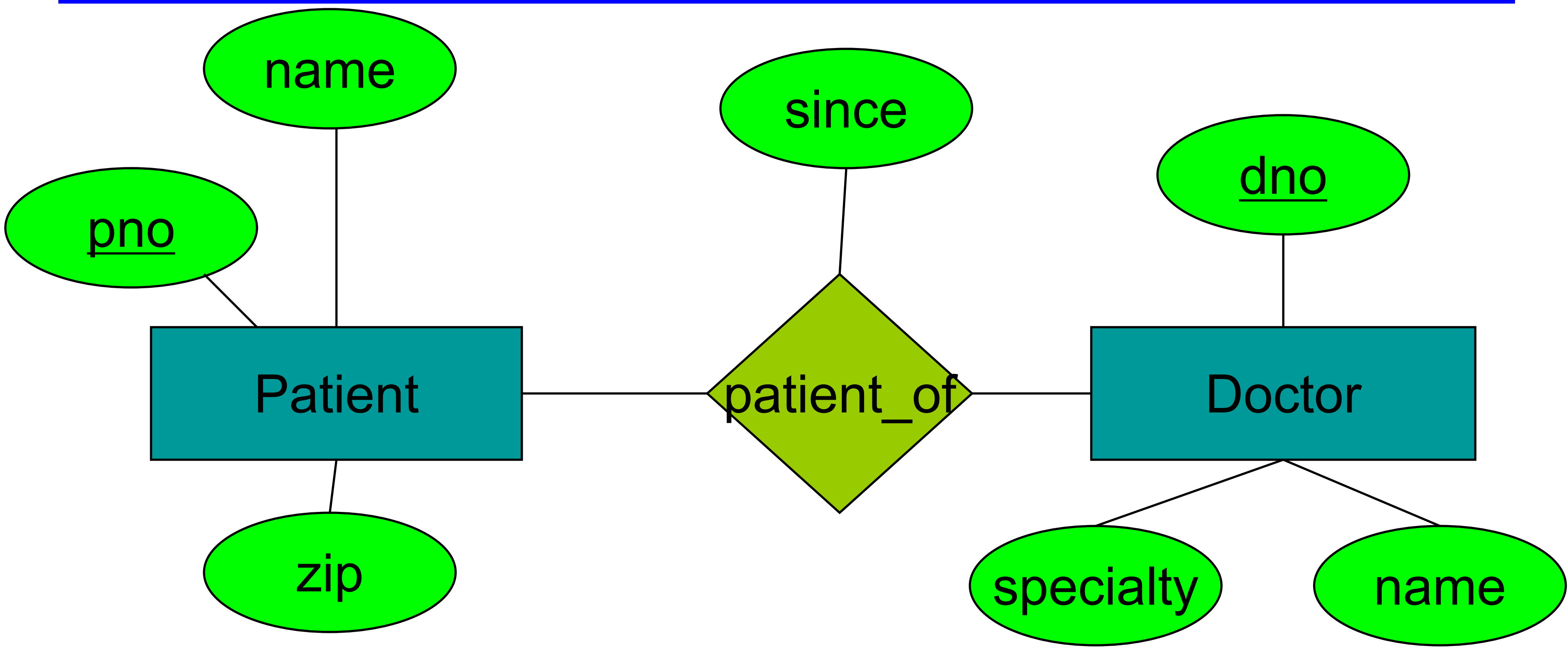


Relational Model:
plus FD's
(FD = functional dependency)



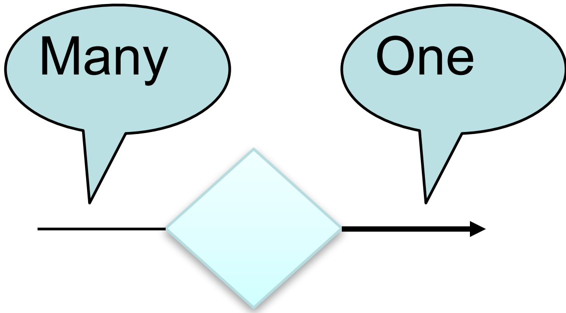
Normalization:
Eliminates anomalies

Entity-Relationship Diagram



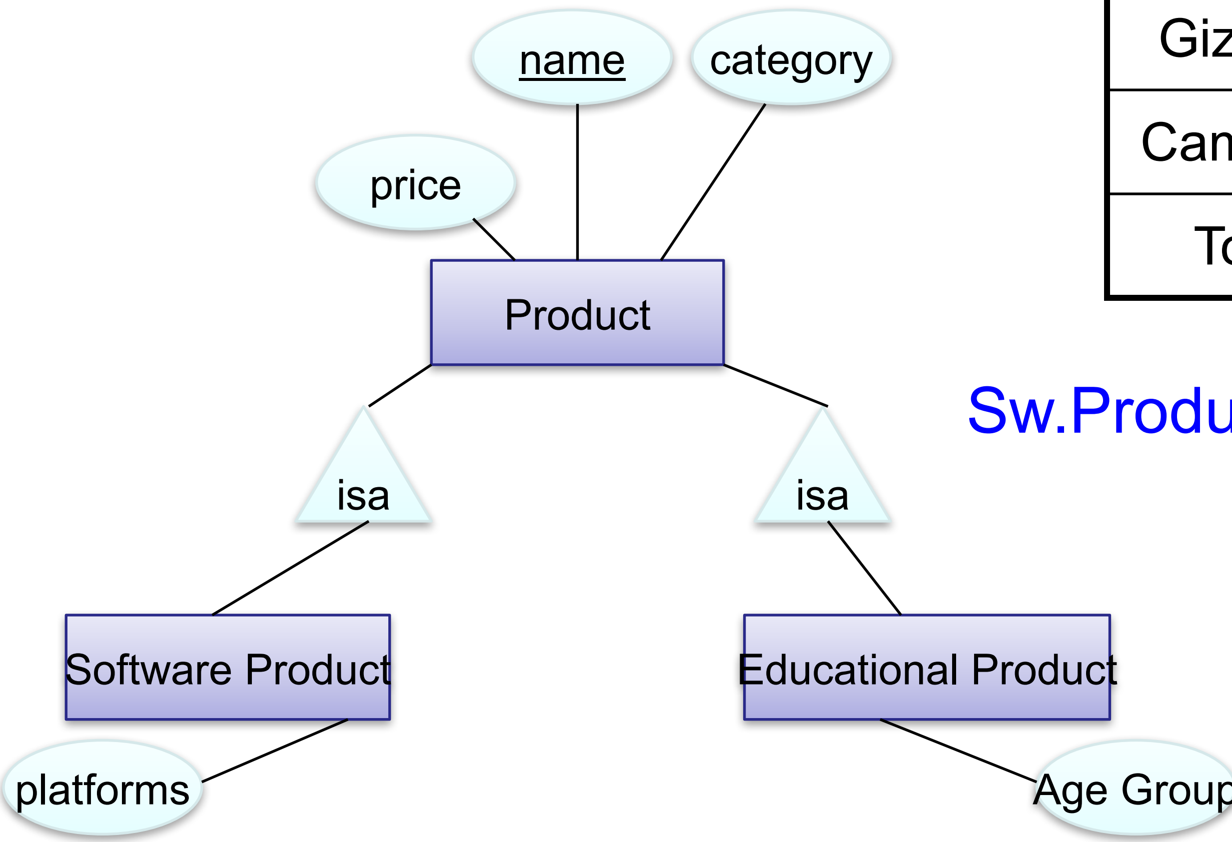
Entity-Relationship Model

- Typically, each entity has a key
- ER relationships can include multiplicity
  - One-to-one, one-to-many, etc.
  - Indicated with arrows



- Can model multi-way relationships
- Can model subclasses
- And more...

Subclasses to Relations



Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	toddler
Toy	retired

Other ways to convert are possible

General approach to Translating Diagram into Relations

Normally translate as follows:

- Each entity set becomes a relation
- Each relationship set becomes a relation
  - Except many-one relationships. Can combine them with entity set.

One ***bad way*** to translate our diagram into relations

- PatientOf (<u>pno</u>, name, zip, dno, since)
- Doctor (<u>dno</u>, dname, specialty)

Outline

- Conceptual db design: entity-relationship model
- Problematic database designs
- Functional dependencies
- Normal forms and schema normalization

Problematic Designs

- Some db designs lead to redundancy
  - Same information stored multiple times
- Problems
  - Redundant storage
  - Update anomalies
  - Insertion anomalies
  - Deletion anomalies

Problem Examples

PatientOf

pno	name	zip	dno	since
1	p1	98125	2	2000
1	p1	98125	3	2003
2	p2	98112	1	2002
3	p1	98143	1	1985

Redundant

If we update to 98119, we get inconsistency

What if we want to insert a patient without a doctor?
What if we want to delete the last doctor for a patient?
Illegal as (pno,dno) is the primary key, cannot have nulls

Solution: Decomposition

Patient

pno	name	zip
1	p1	98125
2	p2	98112
3	p1	98143

PatientOf

pno	dno	since
1	2	2000
1	3	2003
2	1	2002
3	1	1985

Decomposition solves the problem,
but need to be careful…

Lossy Decomposition

Patient

pno	name	zip
1	p1	98125
2	p2	98112
3	p1	98143

PatientOf

name	dno	since
p1	2	2000
p1	3	2003
p2	1	2002
p1	1	1985

Decomposition can cause us to lose information!

Schema Refinement Challenges

- How do we know that we should decompose a relation?
  - Functional dependencies
  - Normal forms

- How do we make sure decomposition does not lose info?
  - Lossless-join decompositions
  - Dependency-preserving decompositions

Outline

- Conceptual db design: entity-relationship model
- Problematic database designs
- Functional dependencies
- Normal forms and schema normalization

Functional Dependency

- A functional dependency (FD) is an integrity constraint that generalizes the concept of a key

- An instance of relation R satisfies the **FD: X $\rightarrow$ Y**
  - if for every pair of tuples t1 and t2
  - if t1.X = t2.X then t1.Y = t2.Y
  - where X, Y are two nonempty sets of attributes in R
- We say that **X determines Y**

- FDs come from domain knowledge

FD Example

An FD holds, or does not hold on an instance:

EmpID	**Name**	**Phone**	**Position**
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

FD Terminology

- FD's are constraints
  - On some instances they hold
  - On others they do not

- If every instance of R will be one in which a given FD will hold, then we say that R satisfies the FD
  - If we say that R satisfies an FD F, we are stating a constraint on R

- FDs come from domain knowledge

Decomposition Problems

- FDs will help us identify possible redundancy
  - Identify redundancy and split relations to avoid it.
- Can we get the data back correctly ?
  - **Lossless-join decomposition**
- Can we recover the FD's on the 'big' table from the FD's on the small tables?
  - **Dependency-preserving decomposition**
  - So that we can enforce all FDs without performing joins

Outline

- Conceptual db design: entity-relationship model
- Problematic database designs
- Functional dependencies
- Normal forms and schema normalization

Normal Forms

- Based on Functional Dependencies
  - 2nd Normal Form (obsolete)
  - **3rd Normal Form**
  - **Boyce Codd Normal Form (BCNF)**

We only discuss these two

- Based on Multivalued Dependencies
  - 4th Normal Form
- Based on Join Dependencies
  - 5th Normal Form

BCNF

A simple condition for removing anomalies from relations:

A relation R is in BCNF if:

If $A_1, ..., A_n \rightarrow B$ is a non-trivial dependency in R ,

then $\{A_1, ..., A_n\}$ is a superkey for R

BCNF ensures that no redundancy can be detected using FD information alone

Our Example

PatientOf

pno	name	zip	dno	since
1	p1	98125	2	2000
1	p1	98125	3	2003
2	p2	98112	1	2002
3	p1	98143	1	1985

pno,dno is a key, but pno $\rightarrow$ name, zip
BCNF violation so we decompose

Decomposition in General

$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)$

$R_1(A_1, ..., A_n, B_1, ..., B_m)$ $R_2(A_1, ..., A_n, C_1, ..., C_p)$

R_1 = projection of R on $A_1, ..., A_n, B_1, ..., B_m$
R_2 = projection of R on $A_1, ..., A_n, C_1, ..., C_p$

Theorem If $A_1, ..., A_n \rightarrow B_1, ..., B_m$
Then the decomposition is lossless

Note: don't necessarily need $A_1, ..., A_n \rightarrow C_1, ..., C_p$

BCNF Decomposition Algorithm

Repeat

choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BCNF condition

split R into

$R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, [rest])$

continue with both R1 and R2

Until no more violations

Lossless-join decomposition: Attributes common to R_1 and R_2 must contain a key for either R_1 or R_2

BCNF and Dependencies

Unit	Company	Product

FD's: Unit $\rightarrow$ Company; Company, Product $\rightarrow$ Unit

So, there is a BCNF violation, and we decompose.

BCNF and Dependencies

Unit	Company	Product

FD's: Unit $\rightarrow$ Company; Company, Product $\rightarrow$ Unit

So, there is a BCNF violation, and we decompose.

Unit	Company

Unit $\rightarrow$ Company

Unit	Product

No FDs

In BCNF we lose the FD: Company, Product $\rightarrow$ Unit

3NF

A simple condition for removing anomalies from relations:

A relation R is in 3rd normal form if :

Whenever there is a nontrivial dep. $A_1, A_2, ..., A_n \rightarrow B$ for R,
then $\{A_1, A_2, ..., A_n\}$ is a super-key for R,
or B is part of a key.

3NF Discussion

- 3NF decomposition v.s. BCNF decomposition:
  - Complex: see book

- Tradeoffs
  - BCNF = no anomalies, but may lose some FDs
  - 3NF = keeps all FDs, but may have some anomalies

Summary

- Database design is not trivial
  - Use ER models
  - Translate ER models into relations
  - Normalize to eliminate anomalies

- Normalization tradeoffs
  - BCNF: no anomalies, but may lose some FDs
  - 3NF: keeps all FDs, but may have anomalies
  - Too many small tables affect performance