CSE 544
Principles of Database Management Systems

Magdalena Balazinska

Fall 2007

Lecture 4 - Schema Normalization

References

- R&G Book. **Chapter 19**: "**Schema refinement and normal forms**"

- Also relevant to this lecture. **Chapter 2**: "**Introduction to database design**" and **Chapter 3.5: "Logical database design: ER to relational"**

Outline

- **Finish discussing SQL (from last lecture)**
- **Finish discussing views (from last lecture)**
- **Schema normalization**
  - Conceptual db design: entity-relationship model
  - Problematic database designs
  - Functional dependencies
  - Normal forms

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

Select-Project-Join Query

Product (<u>pname</u>, price, category, manufacturer)
Company (<u>cname</u>, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

Join between Product and Company

Nested Queries

- **Nested query**
  - Query that has another query embedded within it
  - The embedded query is called a **subquery**

- Why do we need them?
  - Enables us to refer to a table that must itself be computed

- Subqueries can appear in
  - WHERE clause (common)
  - FROM clause (less common)
  - HAVING clause (less common)

Subqueries Returning Relations

Company(name, city)
Product(pname, maker)
Purchase(id, product, buyer)

Return cities where one can find companies that manufacture products bought by Joe Blow

```
SELECT  Company.city
FROM    Company
WHERE   Company.name  IN
                (SELECT Product.maker
                  FROM   Purchase , Product
                  WHERE Product.pname=Purchase.product
                      AND Purchase .buyer = ‘Joe Blow‘);
```

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced By "Gizmo-Works"

```
SELECT  pname
FROM    Product
WHERE   price >  ALL (SELECT price
                      FROM    Product
                      WHERE   maker='Gizmo-Works')
```

Correlated Queries

Movie (title, year, director, length)

Find movies whose title appears more than once.

correlation

```
SELECT DISTINCT title
FROM   Movie AS x
WHERE  year <> ANY
                (SELECT  year
                 FROM    Movie
                 WHERE   title =  x.title);
```

Note (1) scope of variables (2) this can still be expressed as single SFW

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT  pname, maker
FROM      Product AS x
WHERE  price > ALL  (SELECT  price
                     FROM    Product AS y
                     WHERE  x.maker = y.maker AND y.year < 1972);
```

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM    Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

Conceptual evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\dots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Read more about it in the book...

Outline

- **Finish discussing SQL (from last lecture)**
- **Finish discussing views (from last lecture)**
- **Schema normalization**
  - Conceptual db design: entity-relationship model
  - Problematic database designs
  - Functional dependencies
  - Normal forms

Physical Independence

- Definition: **Applications are insulated from changes in physical storage details**

- Early models (IMS and CODASYL): No

- Relational model: Yes
  - Yes through set-at-a-time language: algebra or calculus
  - No specification of what storage looks like
  - Administrator can optimize physical layout

Logical Independence

- Definition: **Applications are insulated from changes to logical structure of the data**
- Early models
  - IMS: some logical independence
  - CODASYL: no logical independence
- Relational model
  - Yes through views

Views

- **View is a relation**
- But rows not explicitly stored in the database
- Instead
- **Computed as needed from a view definition**

Example with SQL

Using relations from Lecture 2

- Supplier(sno,sname,scity,sstate)
- Part(pno,pname,psize,pcolor)
- Supply(sno,pno,qty,price)

```
CREATE VIEW Big_Parts
AS
SELECT * FROM Part WHERE psize > 10;
```

Example 2 with SQL

```
CREATE VIEW Supply_Part2 (name,no)
AS
SELECT R.sname, R.sno
FROM Supplier R, Supply S
WHERE R.sno = S.sno AND S.pno=2;
```

Queries Over Views

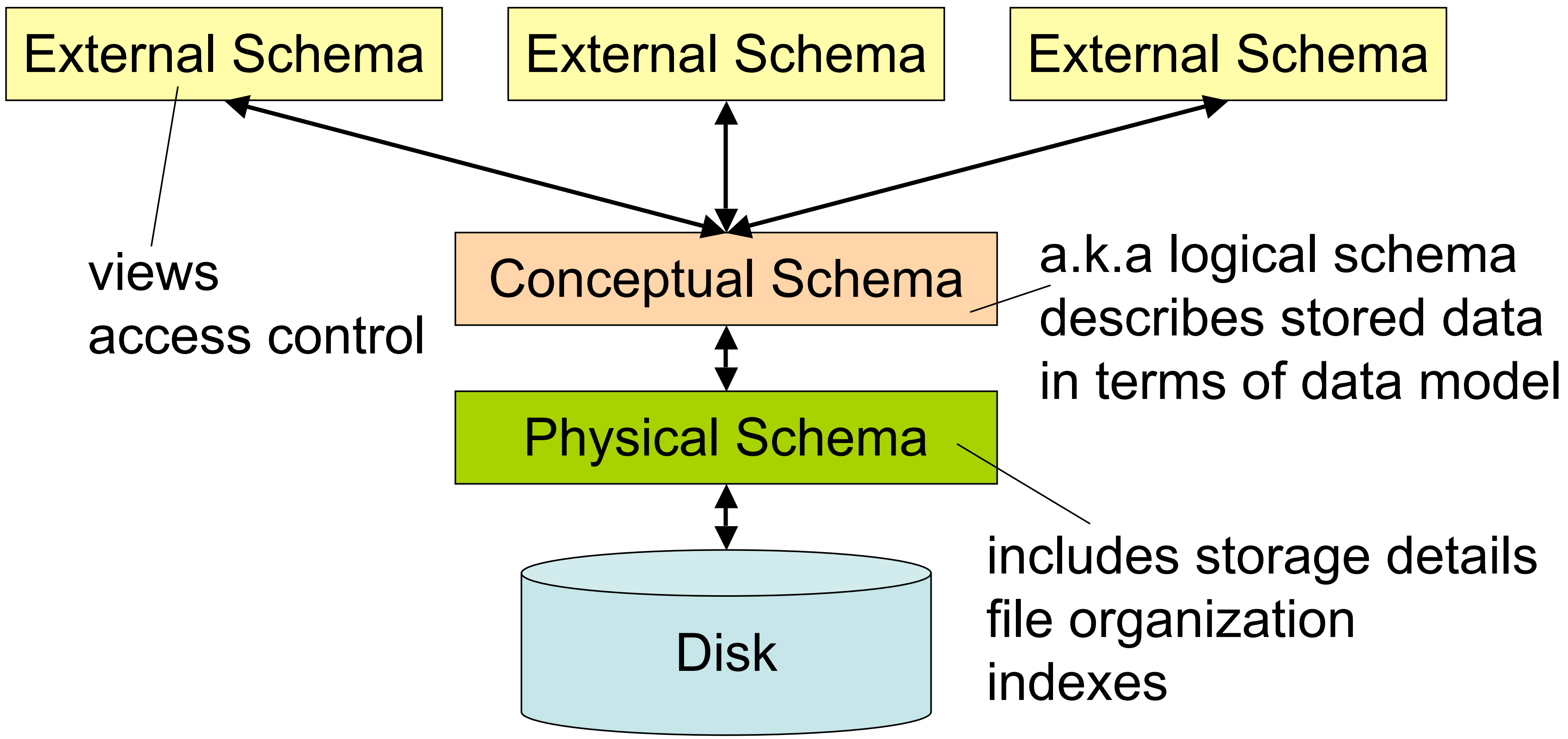
```
SELECT * from Big_Parts
WHERE pcolor='blue';

SELECT name
FROM Supply_Part2
WHERE no=1;
```

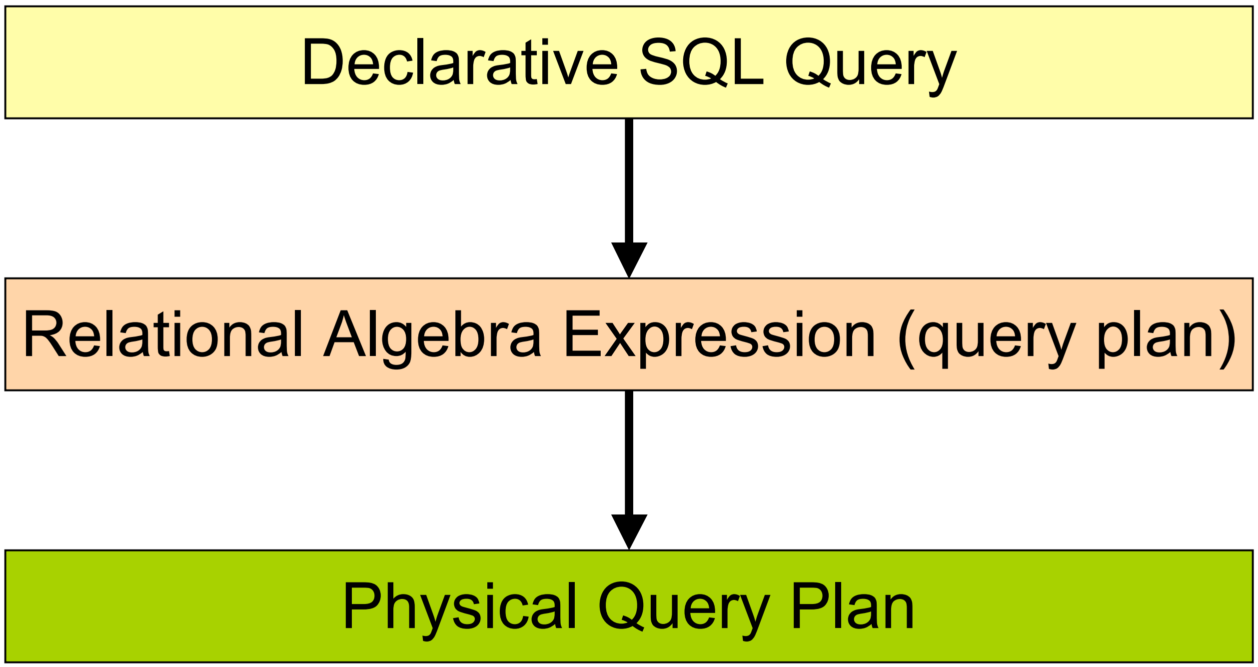
Updating Through Views

- **Updatable views** (SQL-92)
  - Defined on single base relation
  - No aggregation in definition
  - Inserts have NULL values for missing fields
  - Better if view definition includes primary key
- Updatable views (SQL-99)
  - May be defined on multiple tables
- **Messy issue in general**

Levels of Abstraction



Query Translations



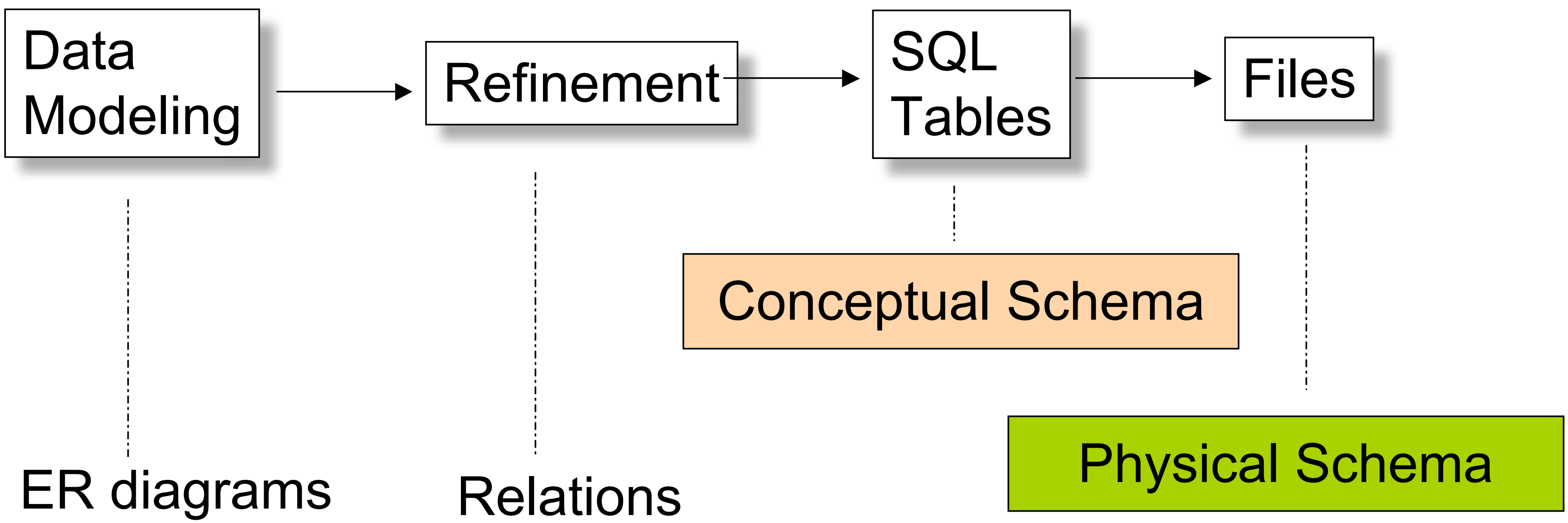
User or application

Optimizer

Outline

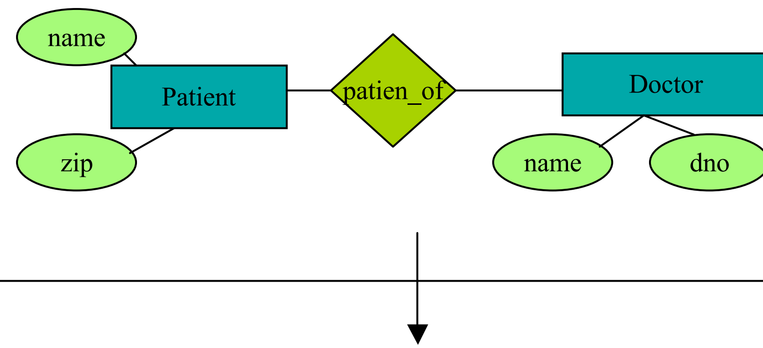
- **Finish discussing SQL (from last lecture)**
- **Finish discussing views (from last lecture)**
- **Schema normalization**
  - Conceptual db design: entity-relationship model
  - Problematic database designs
  - Functional dependencies
  - Normal forms

Database Design Process

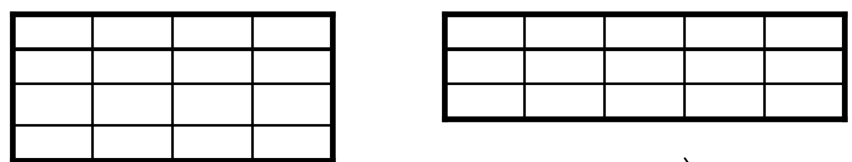


Conceptual Schema Design

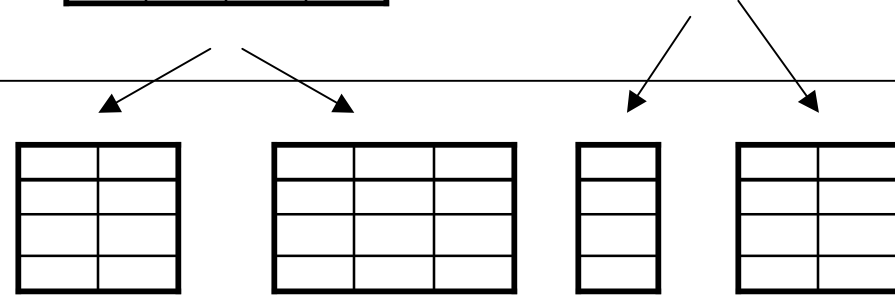
Conceptual Model:



Relational Model:
plus FD's

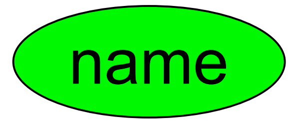


Normalization:
Eliminates anomalies

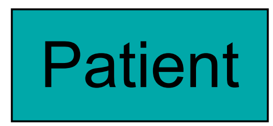


Entity-Relationship Diagrams

Attributes

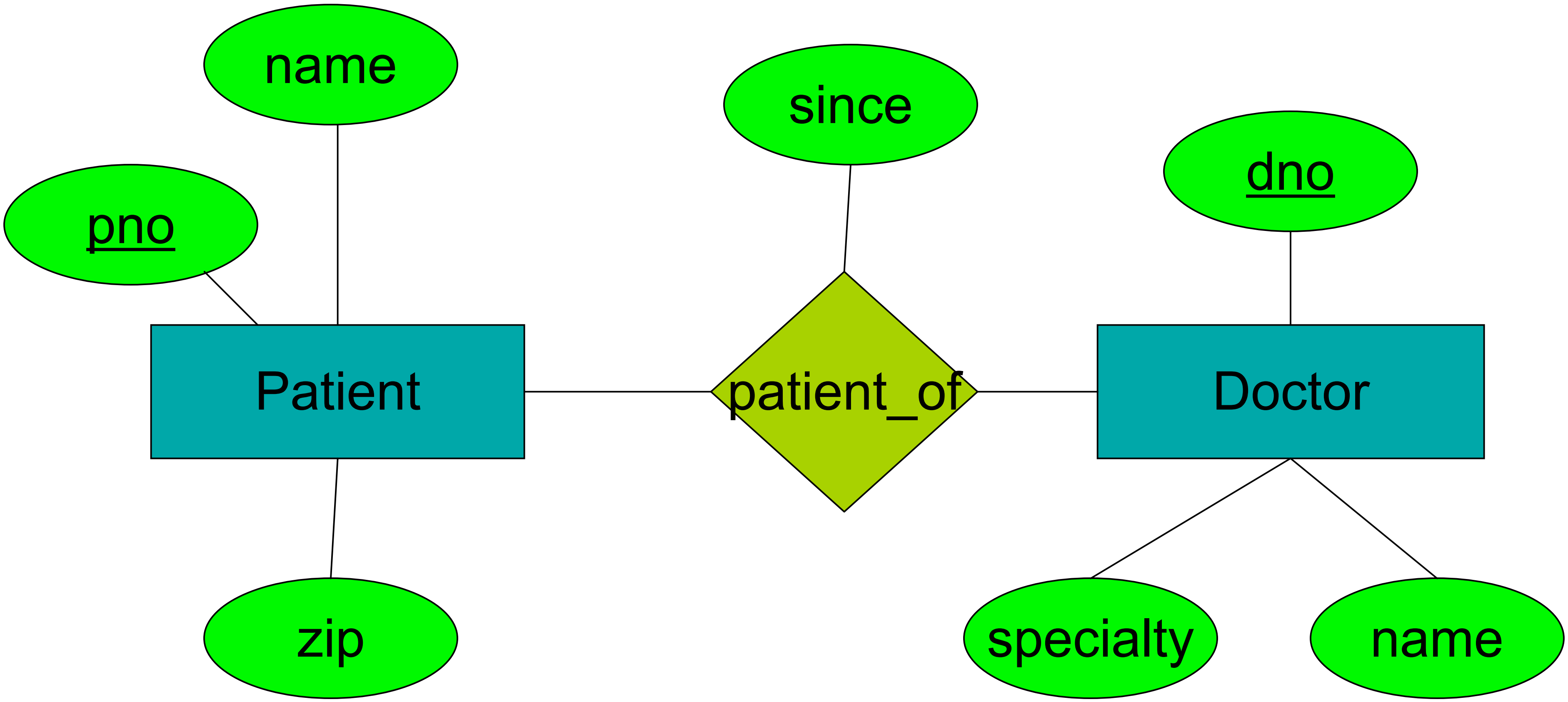


Entity sets



Relationship sets

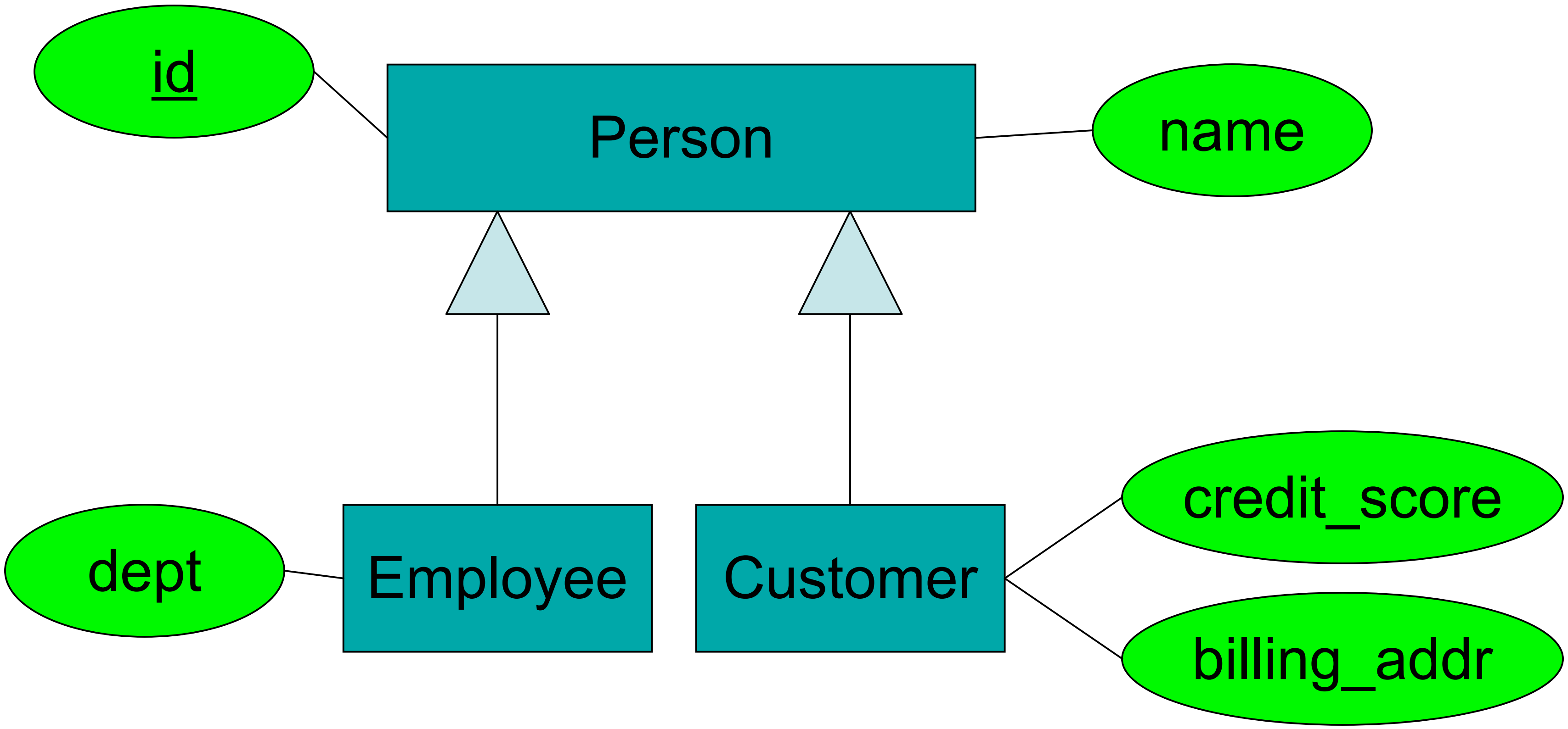
Example ER Diagram



Entity-Relationship Model

- 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...

Example with Inheritance



Example from Phil Bernstein's SIGMOD'07 keynote talk

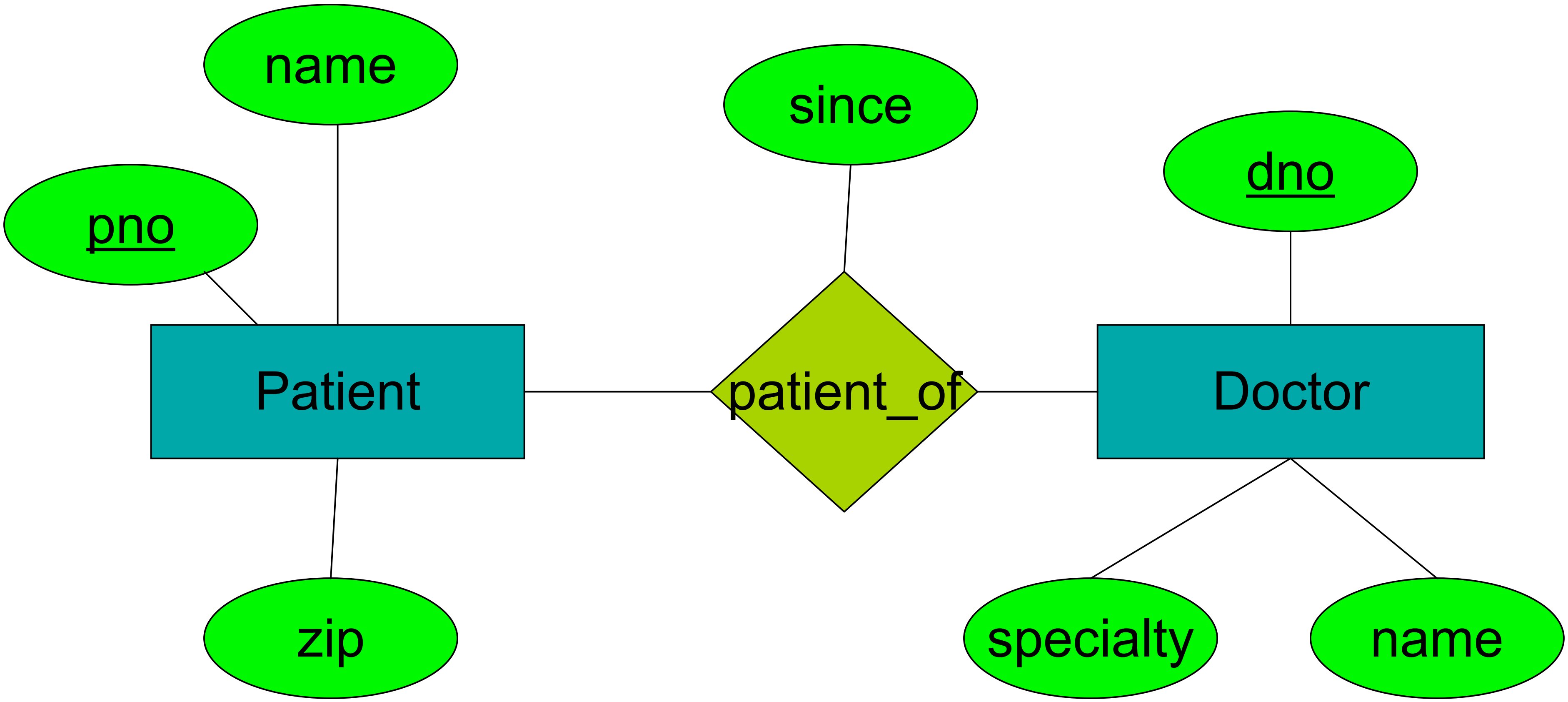
Converting into Relations

- One way to translate our ER diagram into relations
  - HR (id, name)
  - Empl (id, dept) and id is also a foreign key referencing HR
  - Client (id, name, credit_score, billing_addr)

- Today, we only talk about using ER diagrams to help us design the conceptual schema of a database

- In general, apps may need to operate on a view of the data closer to ER model (e.g., OO view of data) while db contains relations
  - Need to translate between objects and relations
  - Can be hard → model management problem

Back to Our Simpler Example



Resulting Relations

- One way to translate diagram into relations

- **PatientOf (pno, name, zip, dno, since)**
- **Doctor (dno, dname, specialty)**

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 lost info?
  - Lossless-join decompositions
  - Dependency-preserving decompositions

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**

Closure of FDs

- Some FDs imply others
- For example: Employee(ssn,position,salary)
  - FD1: ssn → position and FD2: position → salary
  - Imply FD3: ssn → salary

- Can compute **closure** of a set of FDs
- **Armstrong's Axioms**: **sound** and **complete**
  - **Reflexivity**: if $X \supseteq Y$ then $X \rightarrow Y$
  - **Augmentation**: if $X \rightarrow Y$ then $XZ \rightarrow YZ$ for any Z
  - **Transitivity**: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Closure of a Set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$, is the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

Closure Algo. (for Attributes)

Start with $X=\{A_1, \ldots, A_n\}$.

Repeat until X doesn't change do:

if $B_1, \ldots, B_n \rightarrow C$ is a FD and
$B_1, \ldots, B_n$ are all in X
then add C to X.

Can use this algorithm to find keys

- Compute X^+ for all sets X
- If X^+ = all attributes, then X is a superkey
- Consider only the minimal superkeys

Closure Example (for Attributes)

Example:

name → color
category → department
color, category → price

Closures:

name$^+$ = {name, color}

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

color$^+$ = {color}

Closure Algo. (for FDs)

Example:

A, B → C
A, D → B
B → D

Step 1: Compute X^+, for every X:

A+ = A, B+ = BD, C+ = C, D+ = D
AB+ = ABCD, AC+ = AC, AD+ = ABCD
ABC+ = ABD+ = ACD^+ = ABCD
BCD^+ = BCD, ABCD+ = ABCD

Step 2: Enumerate all X, output X → X+ - X

AB → CD, AD→BC, ABC → D, ABD → C, ACD → B

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**

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 need necessarily $A_1, ..., A_n \rightarrow C_1, ..., C_p$

BCNF Decomposition Algorithm

<u>Repeat</u>

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

<u>Until</u> 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 → Company; Company, Product → Unit

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

BCNF and Dependencies

Unit	Company	Product

FD's: Unit → Company; Company, Product → Unit
So, there is a BCNF violation, and we decompose.

Unit	Company

Unit → Company

Unit	Product

No FDs

In BCNF we lose the FD: Company, Product → 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:
  - Use same decomposition steps, for a while
  - 3NF may stop decomposing, while BCNF continues

- 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