CSE 562
Database Systems

Database Design

Some slides are based or modified from originals by
Magdalena Balazinska

cse@buffalo

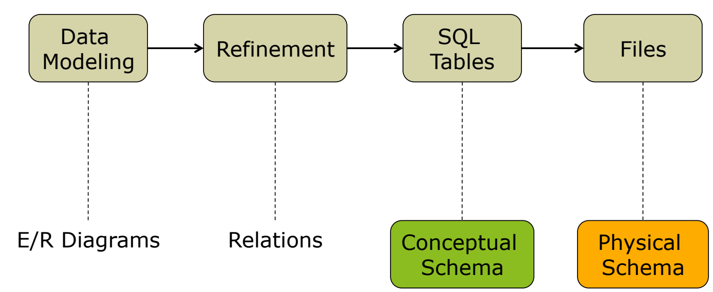
Goal

- **Question**: The relational model is great, but how do I go about designing my database schema?

Outline

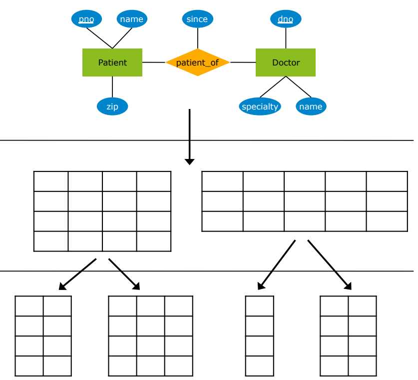
- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization

Database Design Process



Conceptual Schema Design

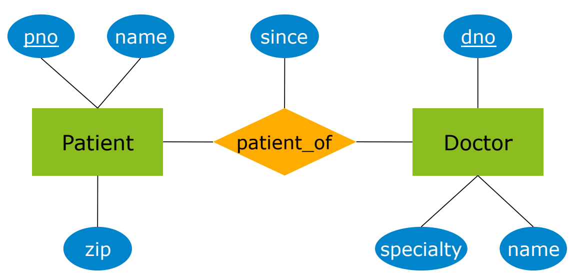
- Conceptual Model
- Relational Model + Functional Dependencies (FDs)
- Normalization Eliminates Anomalies



Entity/Relationship Diagrams

- Attributes
- Entity Sets
- Relationship Sets

Example E/R Diagram



Resulting Relations

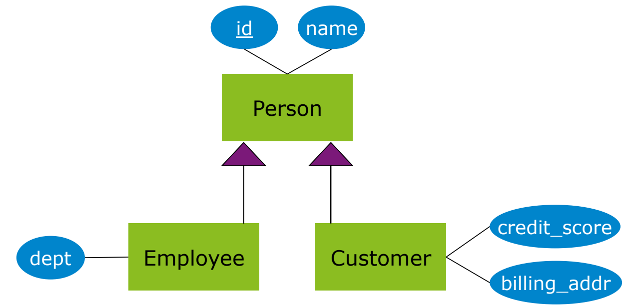
- One way to translate diagram into relations:

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

Entity/Relationship Model

- Typically, each entity has a key
- E/R 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 E/R diagram into relations:
  - HR (id, name)
  - Empl (id, dept) id is also a foreign key referencing HR
  - Client (id, name, credit_score, billing_addr)
- Today, we only talk about using E/R 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 E/R model (e.g., OO view of data) while DB contains relations
  - Need to translate between objects and relations
  - Object-Relational Mapping (ORM)
  - Hibernate, Microsoft ADO.NET Entity Framework, etc.

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

Redundant

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

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 any information?
  - Lossless-join decompositions
  - Dependency-preserving decompositions

Outline

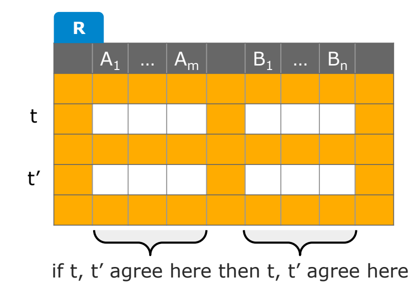
- Conceptual DB Design: Entity/Relationship Model
- Problematic Database Designs
- Functional Dependencies
- Normal Forms and Schema Normalization

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 → Y**
  - if for every pair of tuples t_1 and t_2
  - if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$
  - where X, Y are two nonempty sets of attributes in R
- We say that **X determines Y**
- FDs come from domain knowledge

FD Illustration

The FD $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ holds in R if:
$\forall t, t' \in R$,
$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



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

- FDs are constraints
  - On some instances they hold
  - On others they do not
- If for every instance of R a given FD will hold, then we say that R satisfies the FD
  - If we say that R satisfies an FD, we are stating a constraint on R
- FDs come from domain knowledge

An Interesting Observation

- If all these FDs are true: { **name → color**, **category → department**, **color, category → price** }
- Then this FD also holds: **name, category → price**
- Why ???

How Is This All Useful?

- Anomalies occur when certain "bad" FDs hold
- We know some of the FDs
- Need to find **all** FDs
- Then look for the bad ones

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
  - Set **F+** of all FDs implied by a given set F of FDs
- Armstrong's Axioms: sound and complete
  - Reflexivity: if Y ⊆ X then X → Y
  - Augmentation: if X → Y then XZ → YZ for any Z
  - Transitivity: if X → Y and Y → Z then X → Z
- Convenient split/combine rule:
  If X → Y and X → Z then X → YZ

Example (cont'd)

- Starting from these FDs:
  1. **name → color**
  2. **category → department**
  3. **color, category → price**
- Infer the following FDs:

Inferred FD	Which Rule did we apply?
4. **name, category → name**	
5. **name, category → color**	
6. **name, category → category**	
7. **name, category → color, category**	
8. **name, category → price**	

Example (cont'd)

- Starting from these FDs:
  1. **name → color**
  2. **category → department**
  3. **color, category → price**
- Infer the following FDs:

Inferred FD	Which Rule did we apply?
4. **name, category → name**	Reflexivity
5. **name, category → color**	Transitivity on 4 and 1
6. **name, category → category**	Reflexivity
7. **name, category → color, category**	Split/Combine on 5 and 6
8. **name, category → price**	Transitivity on 7 and 3

- TOO HARD! Let's see an easier way.

Closure of a Set of Attributes

- **Given** a set of attributes $A_1, \ldots, A_n$
- The **closure** $\{A_1, \ldots, A_n\}+$ = the set of attributes B such that $A_1, \ldots, A_n \rightarrow B$
- Example:
  category → department
  name → color
  color, category → price
- Closures:
  name+ = {name, color}
  {name, category}+ = {name, category, color, department, price}
  color+ = {color}

Closure Algorithm For Attributes

To find **closure** $\{A_1, \ldots, A_n\}+$

1. Start with $X=\{A_1, \ldots, A_n\}$
2. Repeat until X doesn't change:
3. if $B_1, \ldots, B_n \rightarrow C$ is a FD and $B_1, \ldots, B_n$ are all in X
4. 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
- Minimal superkeys are keys

Closure For Attributes Example

- Example:
  - category → department
  - name → color
  - color, category → price
- Closures:
  - **name+** = {name, color}
  - **{name, category}+** = {name, category, color, department, price}
  - **color+** = {color}

Another Example

- R(A, B, C, D, E, F)
  - **A, B → C**
  - **A, D → E**
  - **B → D**
  - **A, F → B**
- Compute **{A, B}+**
  - X = {A, B, C, D, E}
- Compute **{A, F}+**
  - X = {A, F, B, C, D, E}

Using Closure To Infer ALL FDs

- Example: **A, B → C**
  - **A, D → B**
  - **B → D**

1. Step 1: Compute X+, for every X:
   A+=A, **B+**=BD, **C+**= C, **D+**= D
   AB+=ABCD, **AC+**=AC, **AD+**=ABCD, **BC+**=BCD, **BD+**=BD, **CD+**=CD
   ABC+=**ABD+**=**ACD+**=**ABCD**, **BCD+**= BCD
   ABCD+=ABCD

2. Step 2: Enumerate all FDs X→Y, s.t. $Y \subseteq X+$ and $X \cap Y = \varnothing$:
   AB→CD, AD→BC, BC→D, 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 FDs on the 'big' table from the FDs 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
  - **3rd Normal Form**
  - **Boyce Codd Normal Form (BCNF)**
- Based on Multi-valued 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 $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency in R, then $\{A_1, \ldots, A_n\}$ is a superkey for R

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

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 → name, zip**
- BCNF violation, so we decompose

Decomposition in General

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

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

- R_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$
- R_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

- **Theorem** If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$, then the decomposition is lossless

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

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 R_1 and R_2
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

- FDs: **Unit → Company**
  Company, Product → Unit

- So there is a BCNF violation, and we decompose

BCNF and Dependencies

Unit	Company	Product

- FDs: **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 dependency $A_1, A_2, \ldots, A_n \rightarrow B$ for R, then $\{A_1, A_2, \ldots, A_n\}$ is a superkey for R, or B is part of a key

3NF Discussion

- 3NF decomposition vs. 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 E/R models
  - Translate E/R 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

This Time

- Design Theory for Relational Databases
  - Chapter 3: 3.1 – 3.5
- High-Level Database Models
  - Chapter 4: 4.1 – 4.6