CSCI-620

Relational Databases

Roadmap

1. Overview of database engines
2. Data modeling
3. Entity-relationship modeling
4. Relational model
5. Modeling hints

Why use a database?

- Useful abstractions
- ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Database Engine

- Two main components
  - Storage manager
    - Authorization/integrity manager
    - Transaction manager
    - File manager
    - Buffer manager
  - Query processor

Storage Manager

- Interface between the DB and the OS
- Responsible for
  - Authorization
  - Interaction with the OS file system
    - Storage access
    - File organization
  - Efficient data storage/modification
    - Indexing and hashing
    - Buffer management

Transaction Manager

- Transaction management
  - Ensures the database is consistent if a failure occurs
  - “All or nothing” (atomic)

- Concurrency control
  - Makes sure multiple operations result in a consistent database

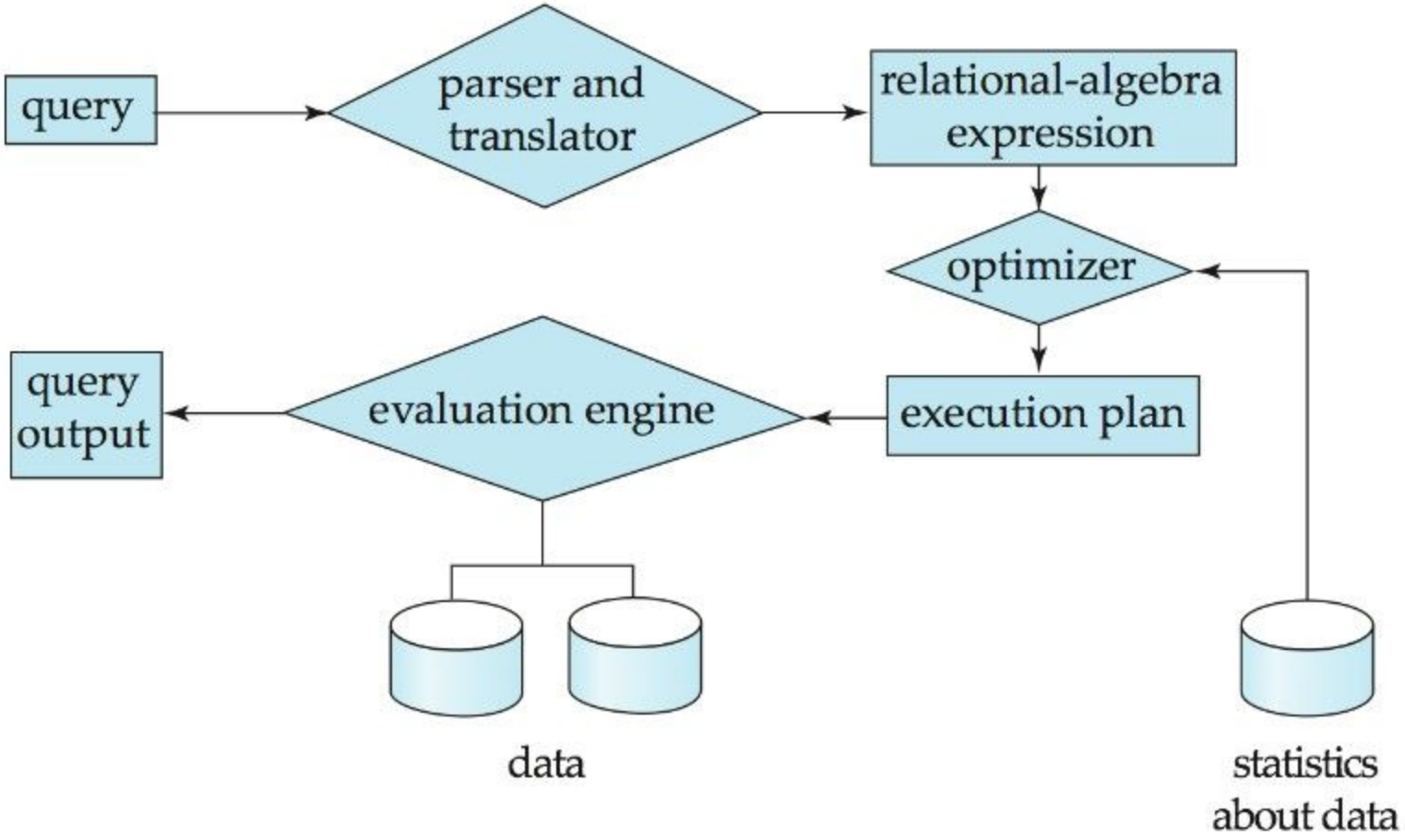
Transaction Manager

- Transactions are collections of operations for a single task

- Example
  - Assume a constraint balance > 0
  - Deduct 50 from A
  - Add 50 to the balance of B
  - Store the new balance

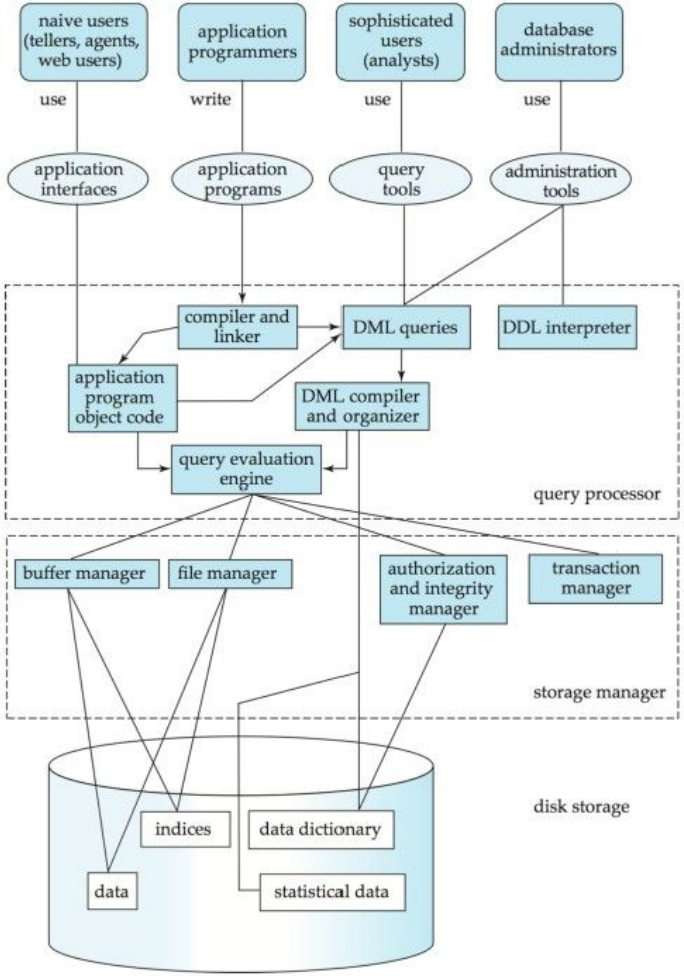
Query Processor

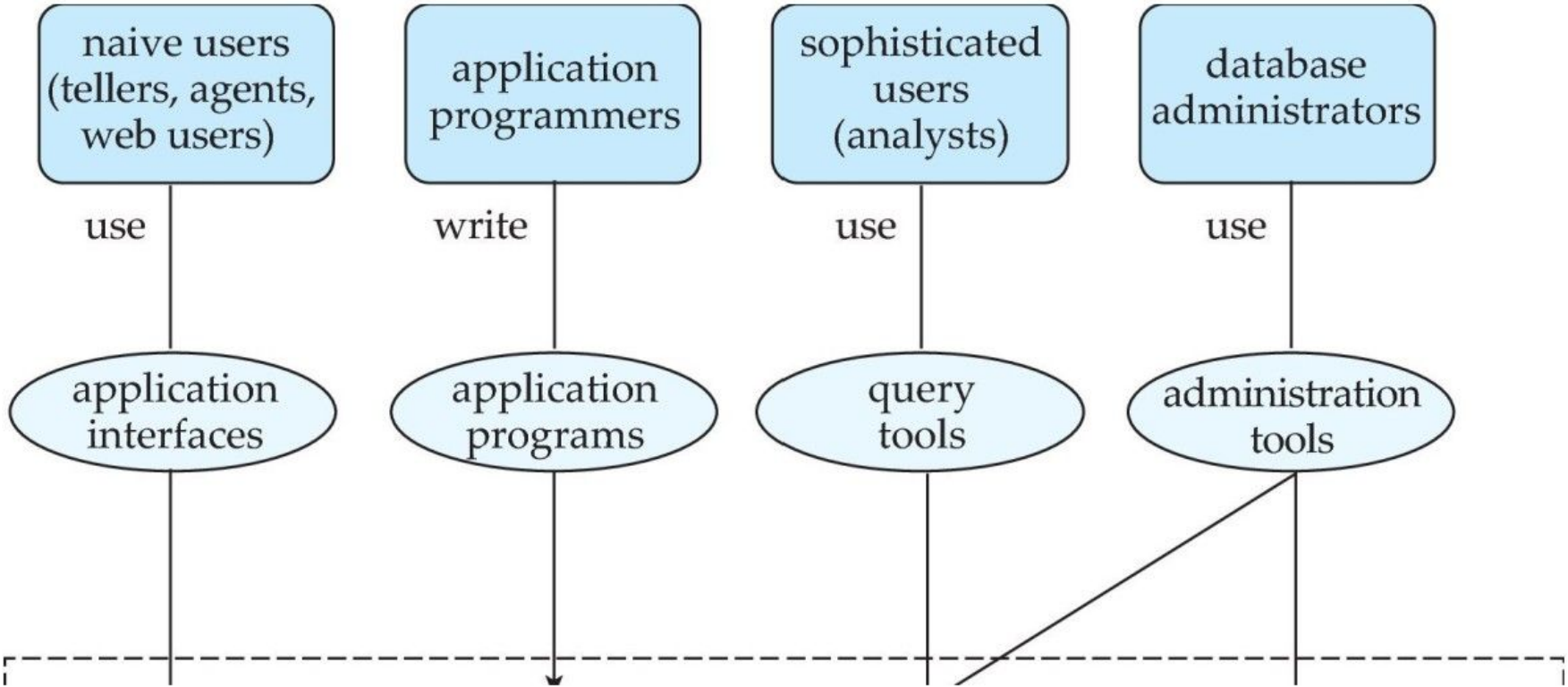
- Three major jobs
  - Parsing and translation
  - Optimization
  - Evaluation


query
parser and translator
relational-algebra expression
optimizer
query output
evaluation engine
execution plan
data
statistics about data

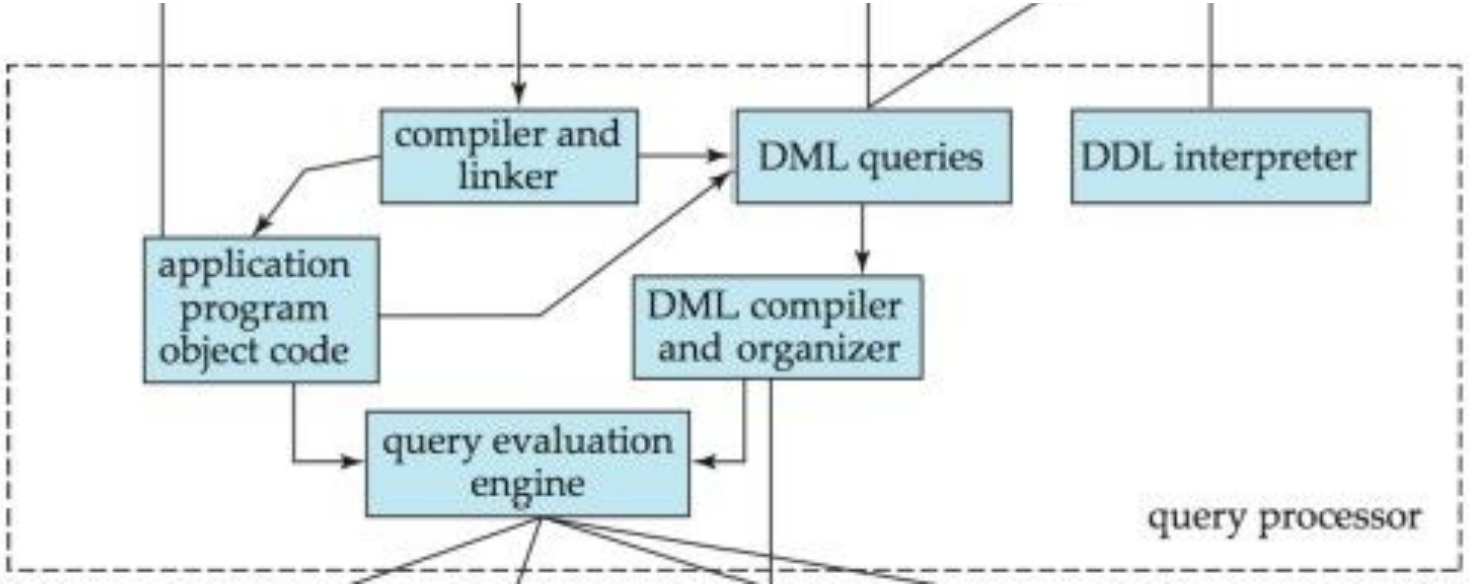
Database Engine

- All components work together
  - Storage manager must make sure transactions are durable
  - Query processor uses indexes managed by the storage manager
  - Transaction manager must provide consistent data to query processor

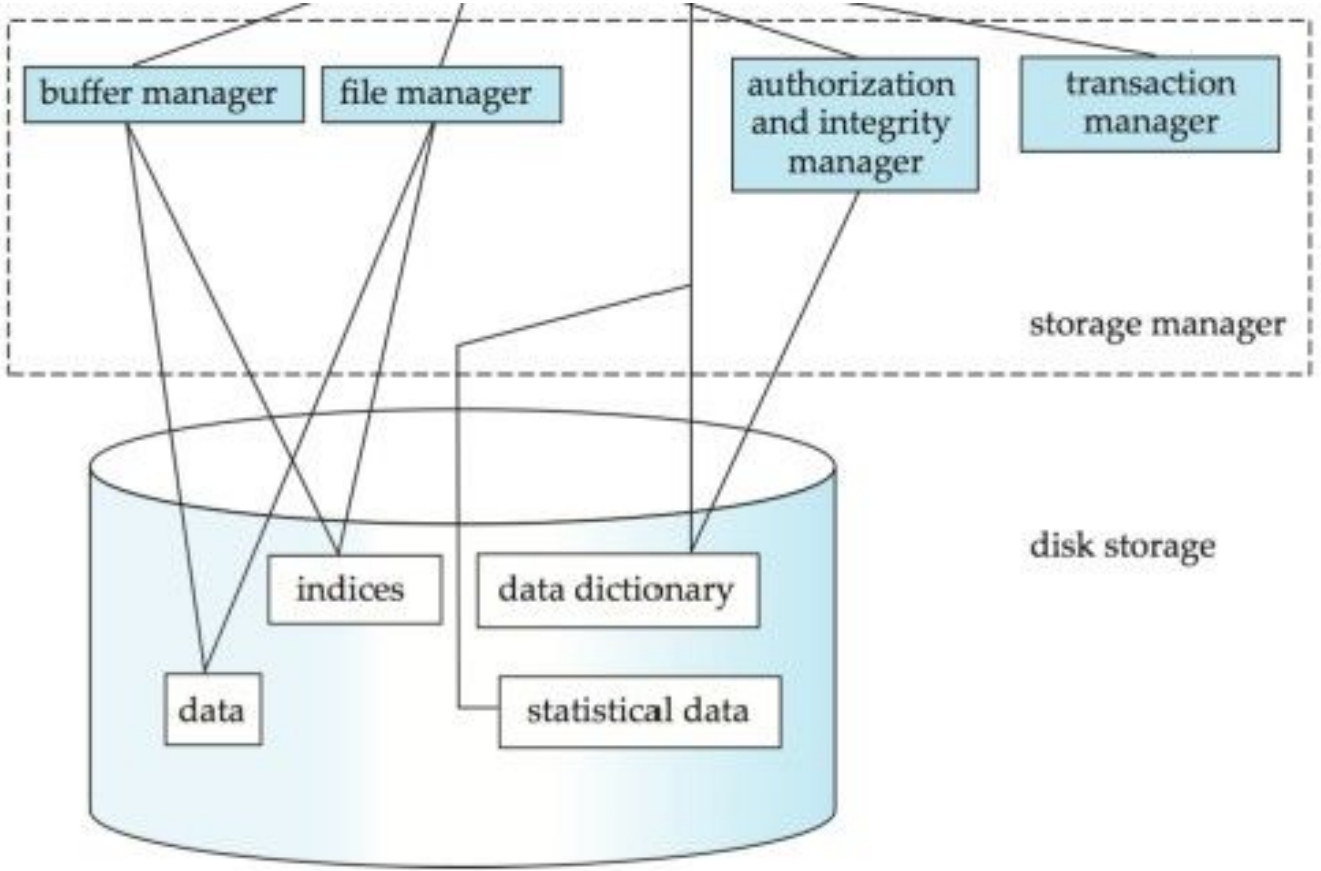

naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data


naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools

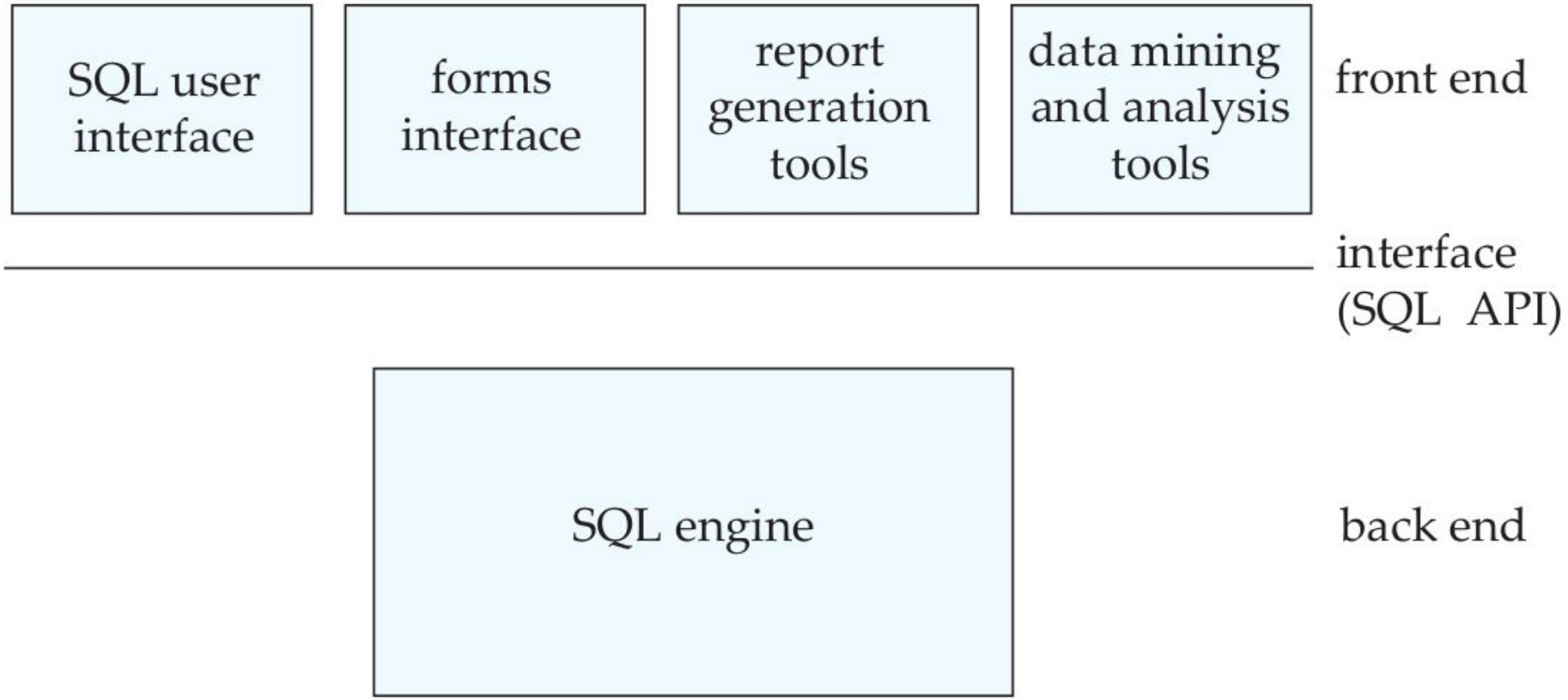
Interface



Query Processor



Storage Manager


SQL user interface
forms interface
report generation tools
data mining and analysis tools
front end
interface (SQL API)
SQL engine
back end

Roadmap

1. Overview of database engines
2. Data modeling
3. Entity-relationship modeling
4. Relational model
5. Modeling hints

Levels of abstraction

1. View Layer

   How applications access data
   (hiding record details, more convenience, etc.)

2. Logical Layer

   How data is stored in the database
   (types of records, relationships, etc.)

3. Physical Layer

   How data is stored on hardware
   (actual bytes, files on disk, etc.)

Data Model

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints

Data Model

- Examples
  - Relational model
  - Entity relationship model
  - Object-based model
  - Semi-structured models (e.g. XML)
  - Network

Database Design

- What makes a *good* design?
- Conceptual modeling (e.g. ER model)
- Normalization theory
  - How do we check for bad designs?
  - How do we fix them?

Conceptual Model

- Before looking at the relational model, we need to have a way to think about what our database needs to store
- Many *conceptual* models exist that are independent of how a particular database stores data
- A common choice is the ER model

Conceptual Model

- Does not specify how data will actually be stored
- Also does not specify the interface we will use to access the data
- Useful for collecting requirements

Database Design

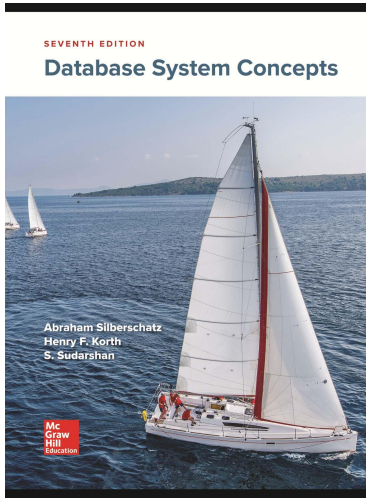
- Steps
  - Analyze requirements
  - Create a conceptual model
  - Create a logical model
  - Create a physical model

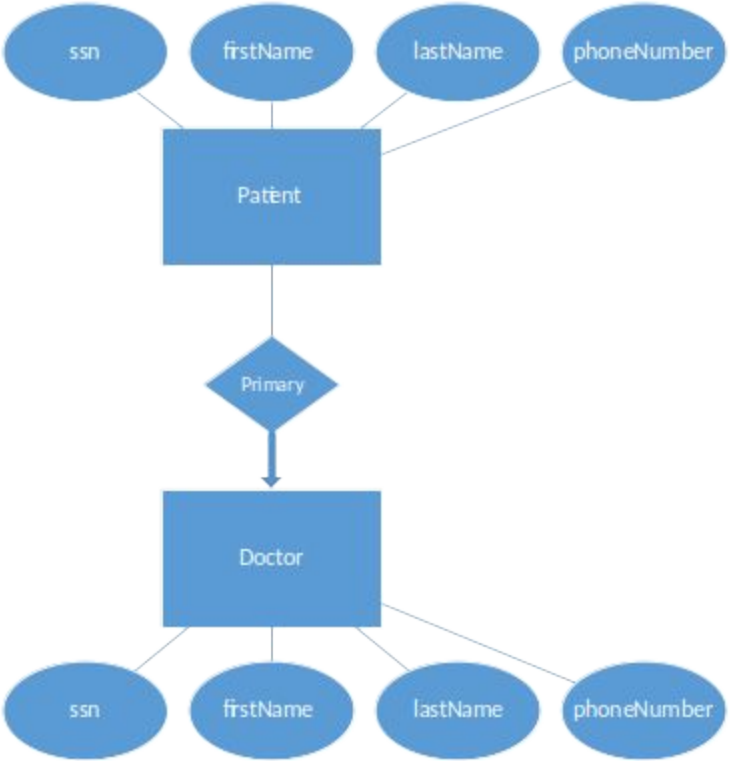
Roadmap

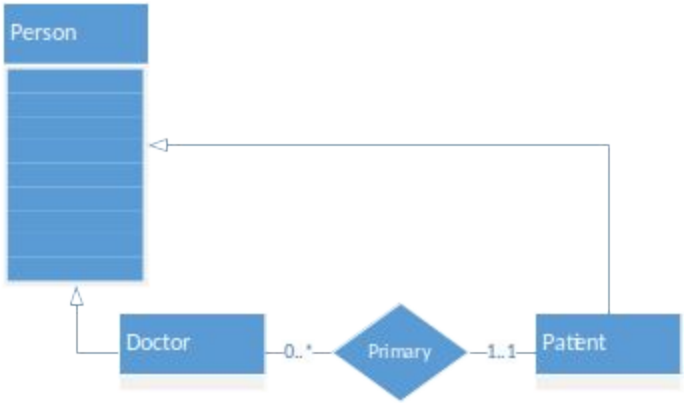
1. Overview of database engines
2. Data modeling
3. Entity-relationship modeling
4. Relational model
5. Modeling hints

Warning

- There are many versions of the ER model we will study
- We'll use the version from the textbook




ssn
firstName
lastName
phoneNumber
Patient
Primary
Doctor
ssn
firstName
lastName
phoneNumber


Person
Doctor
0..*
Primary
1..1
Patient

Notation

Entity

Entity set

Entity Sets

Entity Sets

- A type of thing in the real world
- Distinguishable from other types
- Has a set of properties or attributes possessed by things of the same type

Singular	Plural	Singular	Plural
 Shoe	 Shoes	 Clock	 Clocks
 Ball	 Balls	 Spoon	 Spoons
 Car	 Cars	 Cup	 Cups

Single or plural names?


UNIVERSITY HOSPITAL
expires 09/2004
Pediatric Patient ID
Jacob Jones
Patient ID #: 750101
DOB: 08/01/2003
Sex: M

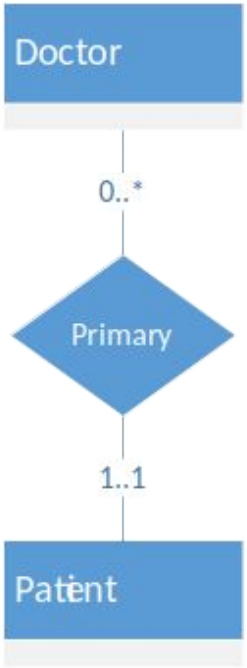

Person
ssn
firstName
middleName
lastName
phoneNumber
birthDate
gender
email
occupation

Attributes

Attributes

- Each attribute has an associated type which is normally *atomic* (indivisible)
- One or more attributes called the *primary key* uniquely identify an entity
- The set of valid values for an attribute is called the *domain*
- Attributes may be *simple* or *complex*


Person
ssn
firstName
middleName
lastName
phoneNumber
birthDate
gender
email
occupation



Relationship

Relationship set

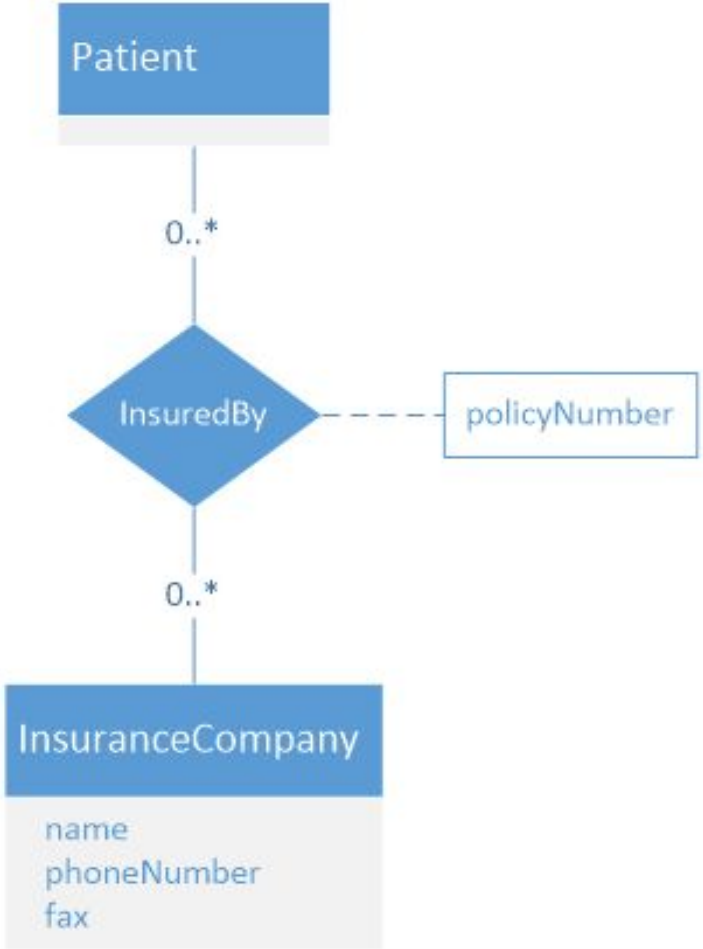
Relationships

Let’s try building a model

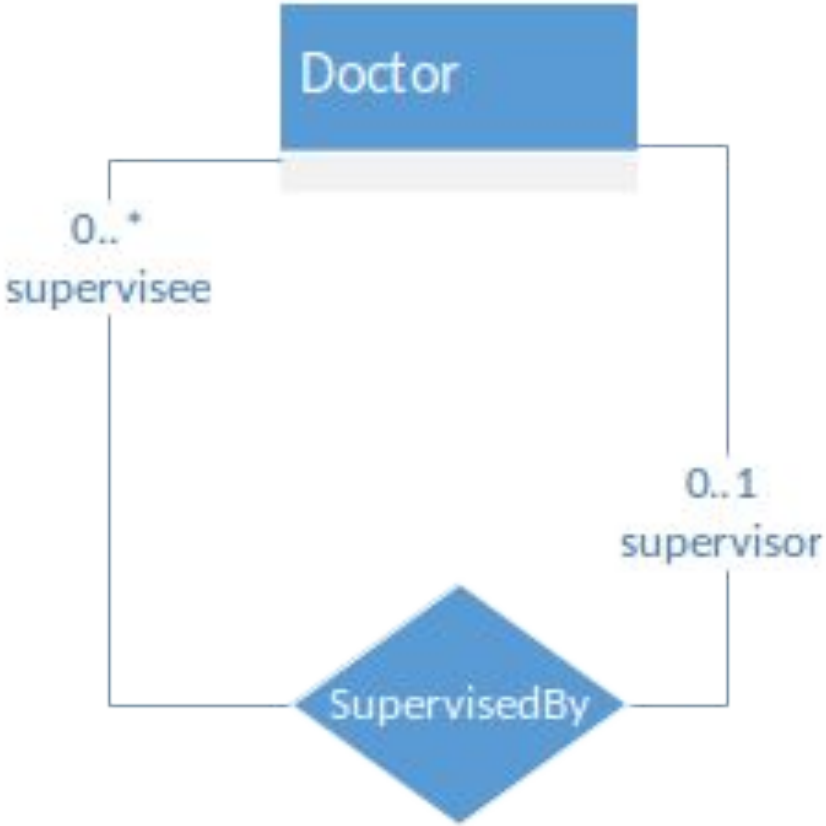
Non-binary Relationships

Relationships

- May also have attributes
- Each relationship has a *cardinality* or a restriction on the number of entities
- Implicitly defines a *role* for each entity set in the relationship



Relationship Set Attributes



Roles

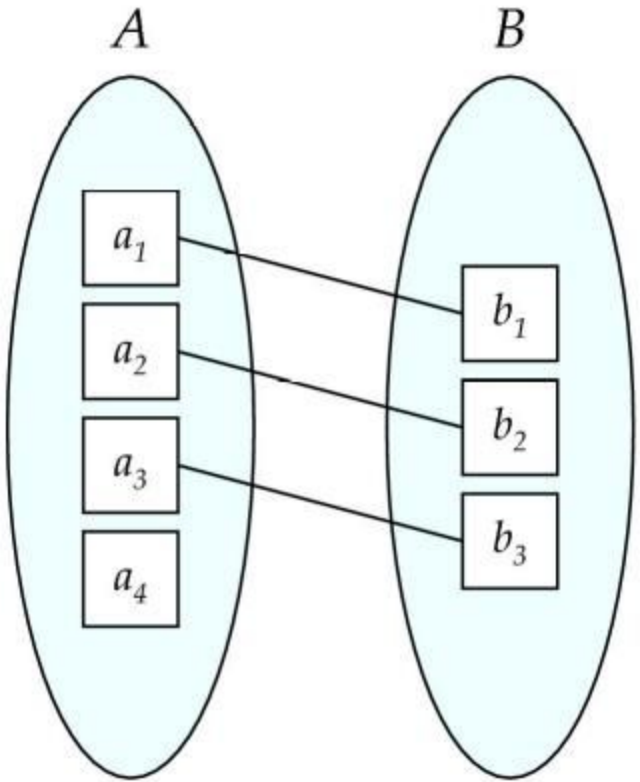
1 to 1

1 to N

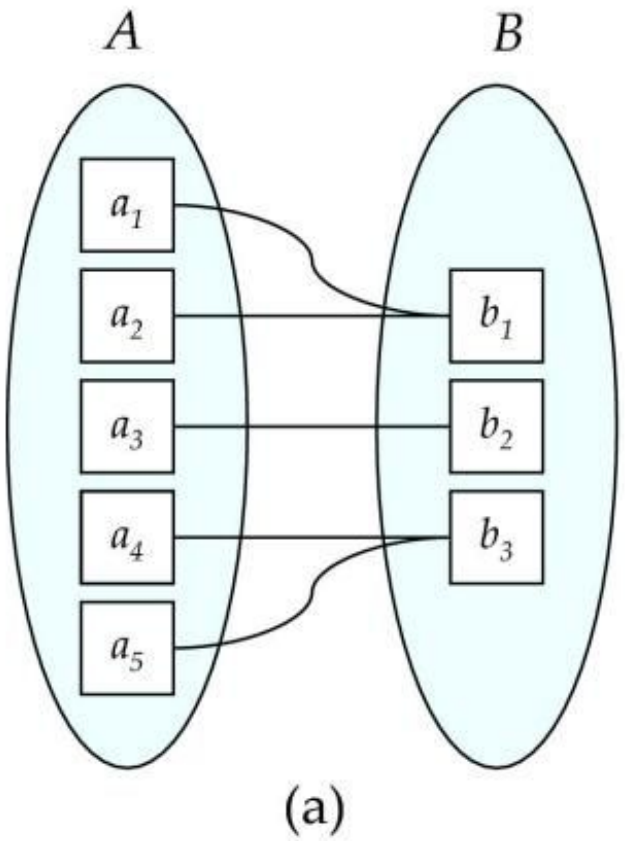
M to N

Cardinalities

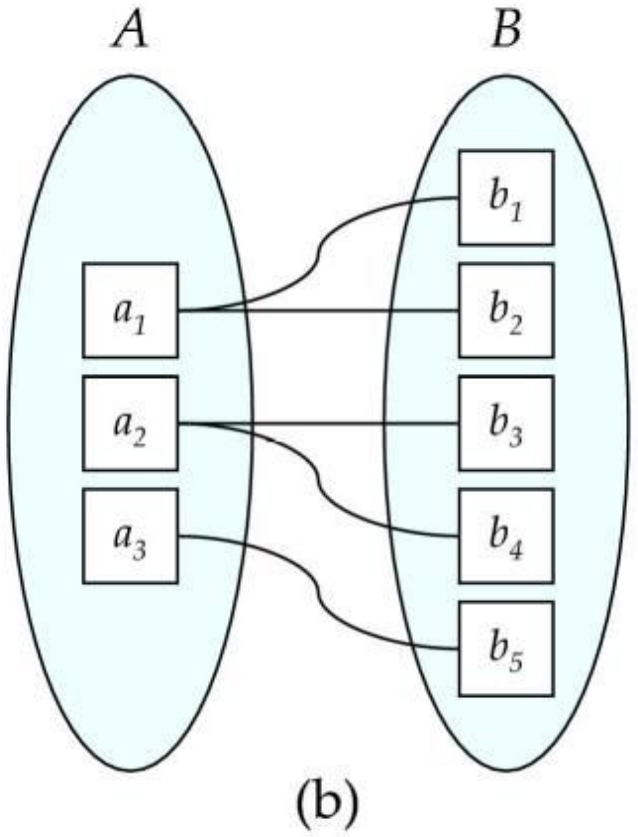

Visit
scheduled
height
weight
bloodPressure
temperature
otherNotes
1..1
Generates
1..1
Bill
billingDate
dueDate
amount


A
B
a_1
a_2
a_3
a_4
b_1
b_2
b_3

One to one

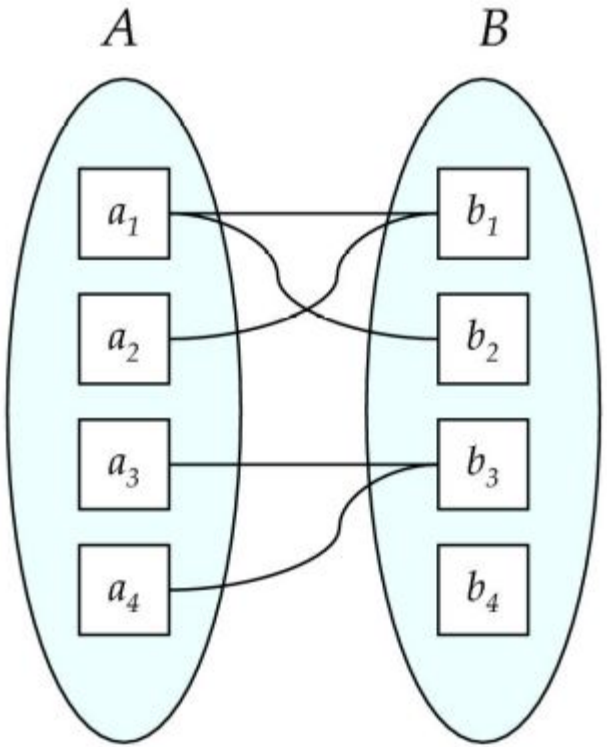


Many to one

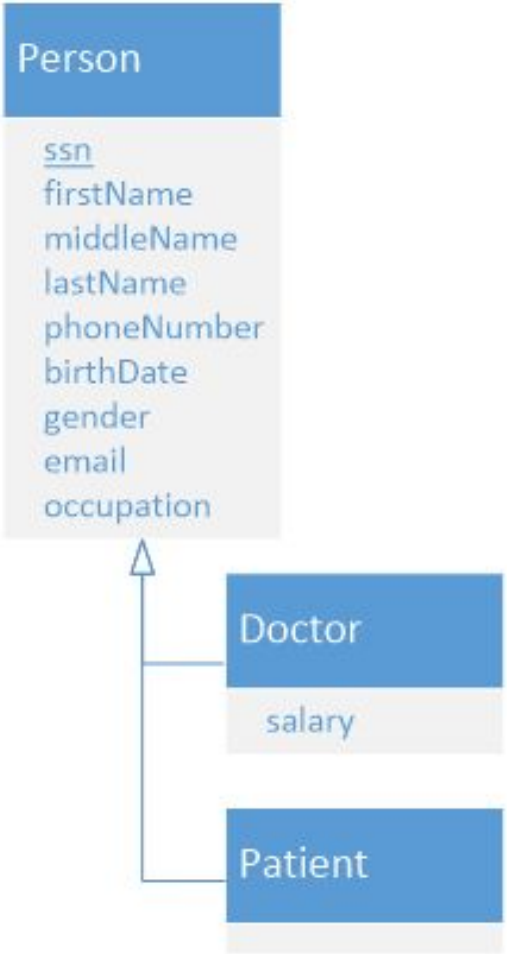


One to many


Bill
billingDate
dueDate
amount
0..*
isPaidBy
1..*
Payment
paymentDate
amount
method


A
B
a_1
a_2
a_3
a_4
b_1
b_2
b_3
b_4

Many to many



Inheritance

Identity

- Each entity in an entity set must have some type of key
- This is usually a subset of the attributes associated with an entity
- The key should (never) or rarely change

Super Candidate Primary

Types of keys

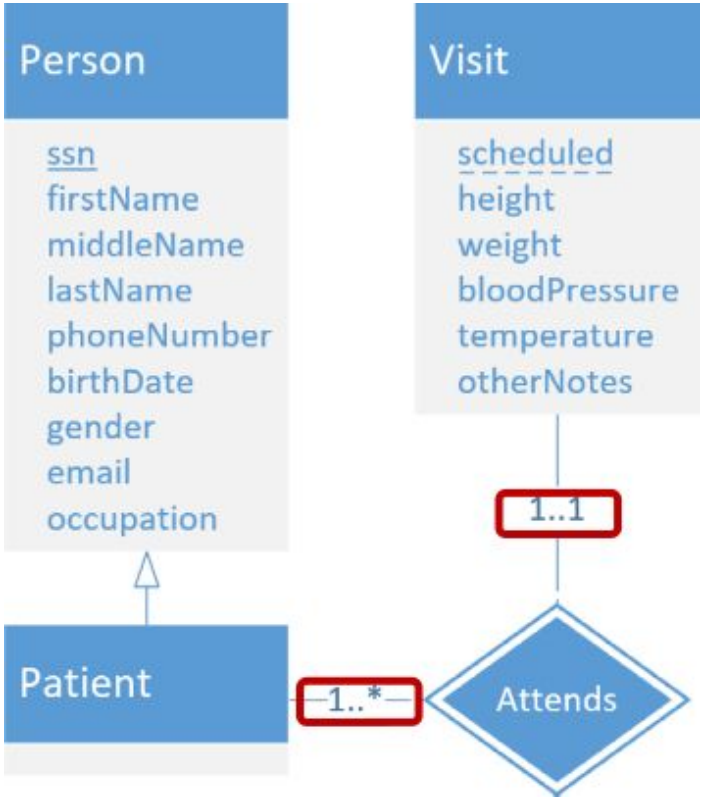
What if there is no candidate key?

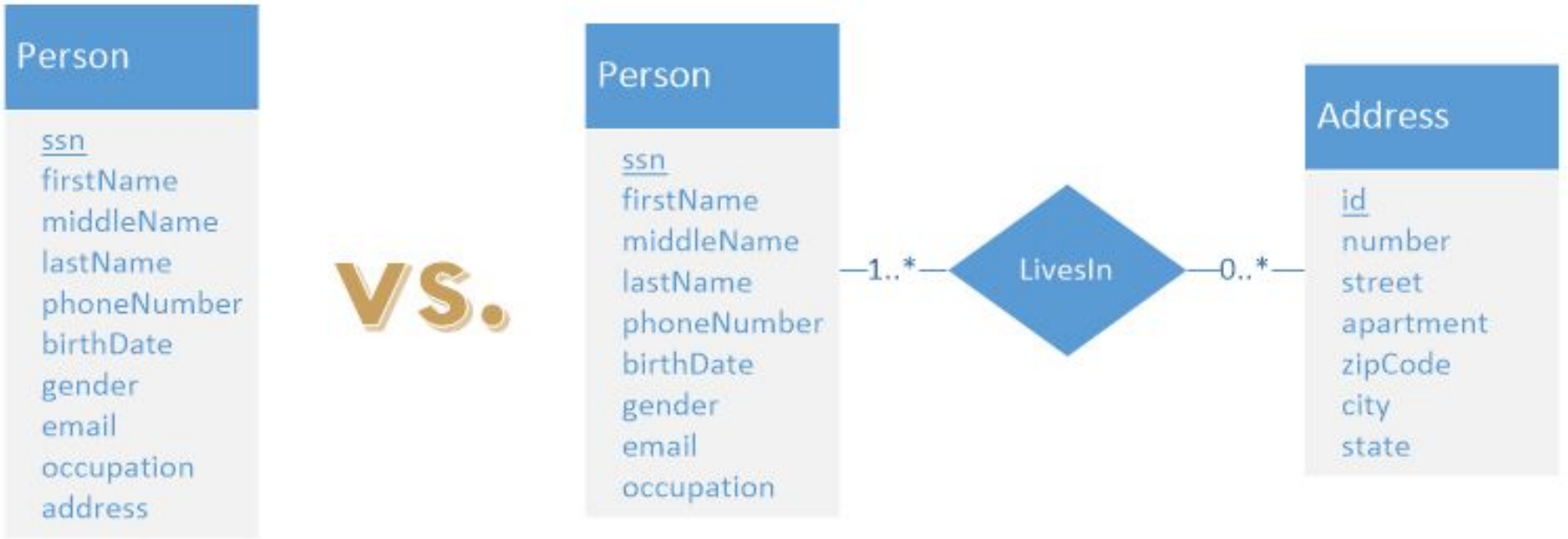
Solution 1

- Create an artificial ID attribute
- Ensure that a unique value is assigned
- Examples
  - Order number
  - Customer ID

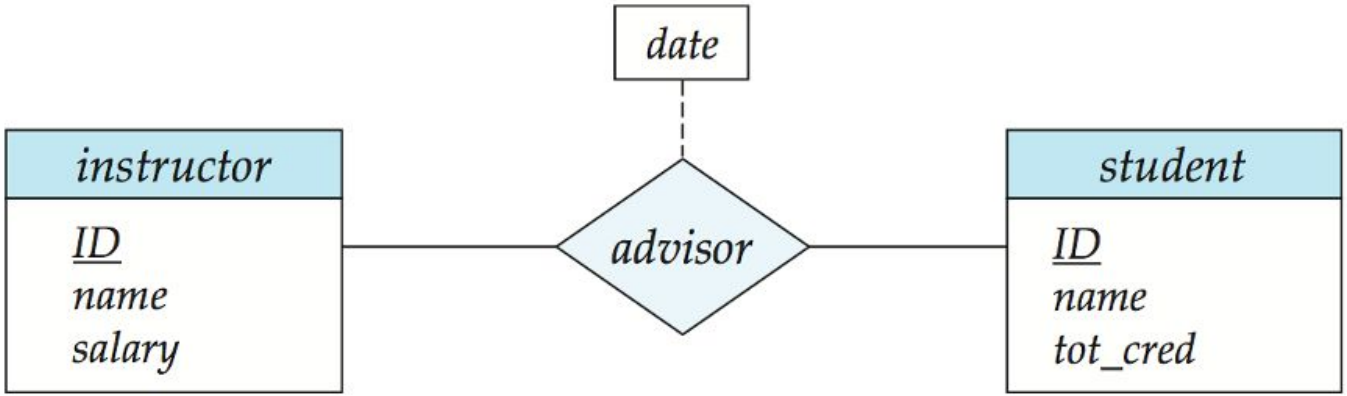
Solution 2

- Use a *weak entity set*





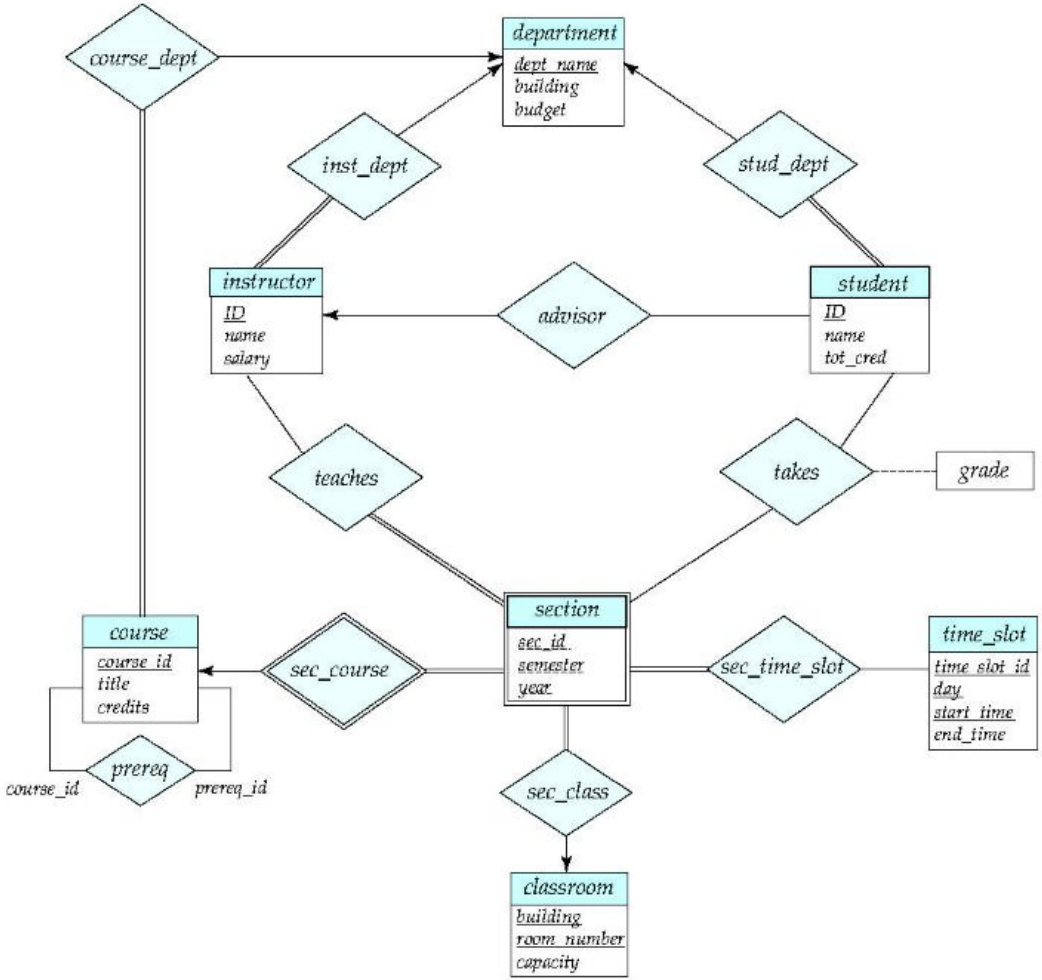
Design Decisions



Design Decisions

Design Decisions

- Attributes or entity sets?
- Entity sets or relationships?
- Strong or weak entity sets?
- Specialization/generalization


department
dept_name
building
budget
course_dept
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
course
course_id
title
credits
sec_course
section
sec_id
semester
year
sec_time_slot
time_slot
time_slot_id
day
start_time
end_time
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity

Reminder

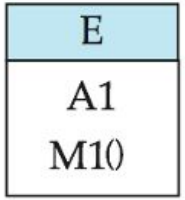
- Remember that the ER model is *conceptual* and not what a database actually uses
- We need a more concrete model to actually implement our application
- Coming up next, the relational model!

Let’s try more modeling!

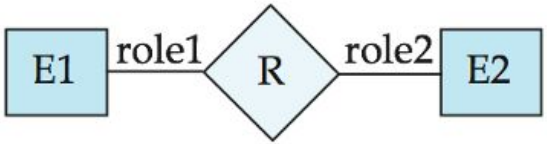
UML

- Unified Modeling Language
- Used to graphically model software systems (*could* replace ER)
- We will use ER for databases, but UML to model your project software

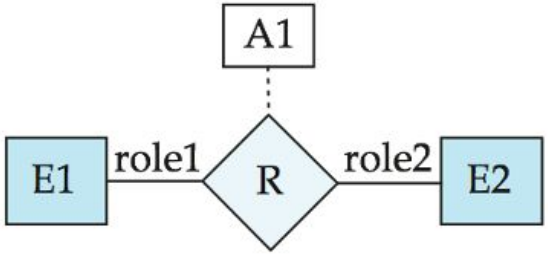
ER Diagram Notation



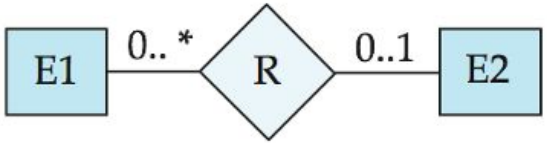
entity with attributes (simple, composite, multivalued, derived)



binary relationship

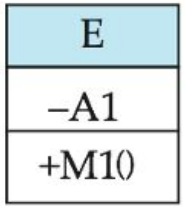


relationship attributes

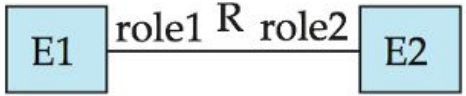


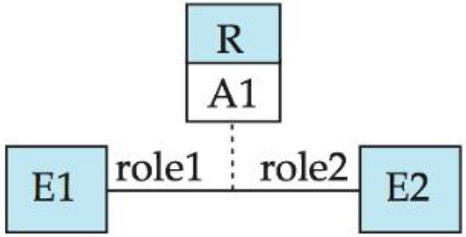
cardinality constraints

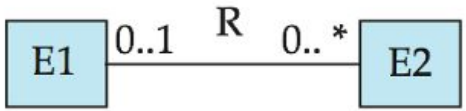
Equivalent in UML



class with simple attributes and methods (attribute prefixes: + = public, – = private, # = protected)

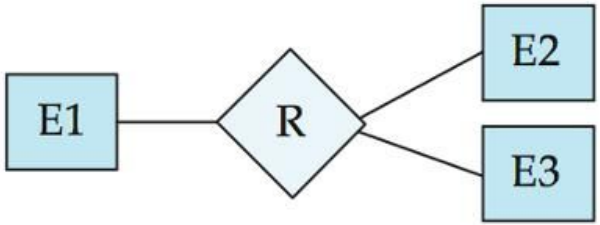




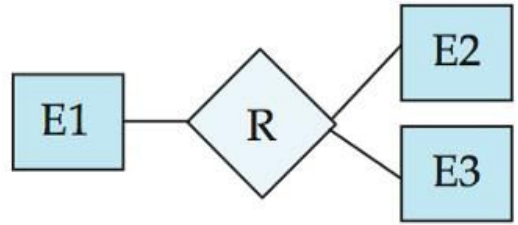


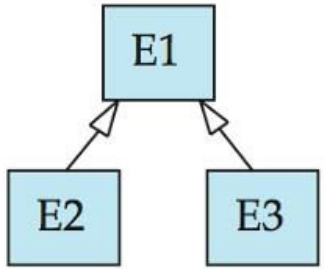
ER Diagram Notation

Equivalent in UML

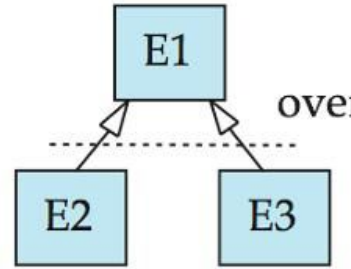


n-ary relationships

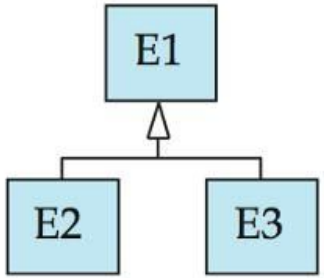




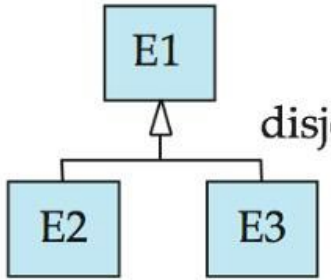
overlapping generalization



overlapping



disjoint generalization



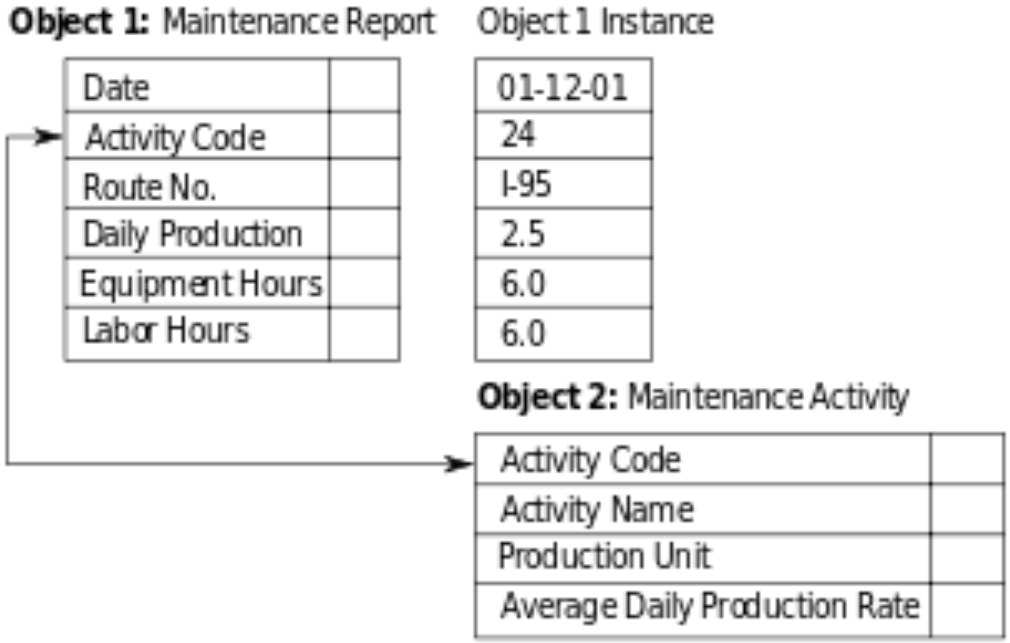
disjoint

Roadmap

1. Overview of database engines
2. Data modeling
3. Entity-relationship modeling
4. Relational model
5. Modeling hints

What now?

- Database systems don't use the ER model directly (it's a *conceptual* model)
- We need to move on to the next stage and pick a *logical* model
- The first model we'll explore is the *relational* model



Object-Relational Models

```
<Steps>
  <UserStep Code="Create" Start="NR" NotAssignedBehaviour="ERR" RequestComment="true"
   Help="true" Delegate="true" Iteration="1" CanSendToPool="true" StepAssignmentType="S">
  <Directions>
    <Direction Code="Direction1" To="Step1" Direction="F">
       <Condition Type="BS" Script="GT(WFDATA( "MAIN", "AMOUNT" ),"1000")" />
       <From>Create</From>
       </Direction>
    <Direction Code=" Direction2" To="Step2" Direction="F">
       <Condition Type="BS" Script="GT(WFDATA( "MAIN", " AMOUNT " ),"5000")" />
       <From>Create</From>
       </Direction>
    <Direction Code=" Direction3" To="Step3" Direction="F">
       <Condition Type="A" />
       <From>Create</From>
       </Direction>
     </Directions>
```

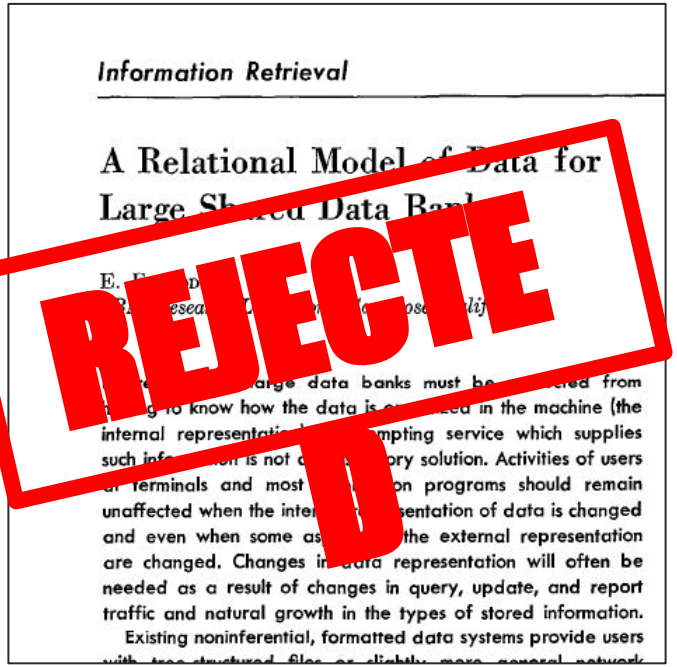
Semi-Structured Models

Patient				
ssn	firstName	middleName	lastName	primaryDoctor
235-14-7854	Sandra	null	Smith	943-23-9874
192-48-0924	John	Richard	Moore	862-74-3611

Doctor		
ssn	firstName	lastName
943-23-9874	Rachel	Wang
862-74-3611	Chris	Patel

Relation instances

Great idea!


Information Retrieval
A Relational Model
Data for
Large
Data
REJECTE
D
data banks must be
from
know how the data is
in the machine (the
internal representati
mpting service which supplies
such inf
is not
ory solution. Activities of users
terminals and most
on programs should remain
unaffected when the inte
sentation of data is changed
and even when some as
the external representation
are changed. Changes in
representation will often be
needed as a result of changes in query, update, and report
traffic and natural growth in the types of stored information.
Existing noninferential, formatted data systems provide users

At first...

Comments

- “… at first sight I doubt that anything complex enough to be of practical interest can be modeled using relations.”
- “… any realistic model might end up requiring dozens of interconnected tables —hardly a practical solution given that, probably, we can represent the same model using two or three properly formatted files.”
- “The paper can be safely rejected.”

However

- “… no real-world example (…) any model of practical interest can be cast in it.”
- “… no experiments (…) how it compares with traditional ones on real-world problems.”
- “… to extract any significant answer from any real database, the user will end up with the very inefficient solution of doing a large number of joins.”

Relations

- A relation is a group of related attributes like in an entity set
- Each relation should have a *primary key*
- Relations may also have *foreign keys* or attributes which refer to other relations
- In a relational database, these are represented as *tables*

Patient

SSN	First Name	Middle Name	Last Name
235-14-7854	Sandra		Smith
192-48-0924	John	Richard	Moore
821-13-2108	Laura		Turner

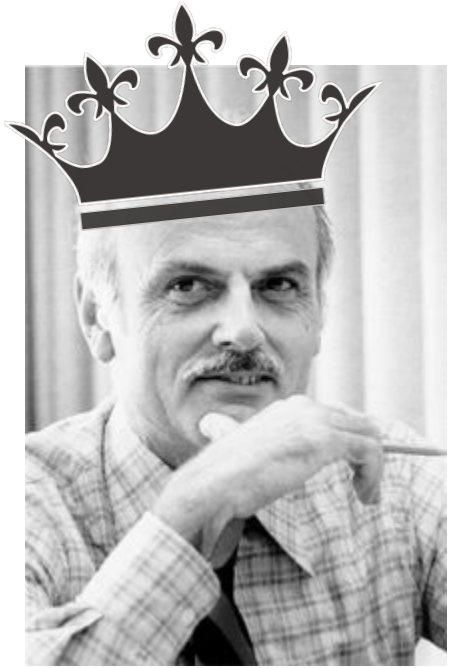
Visit

SSN	Scheduled	Weight
235-14-7854	09/03/2015	141.5
821-13-2108	10/18/2015	167.8

Patient-Visit

SSN	First Name	Middle Name	Last Name	Scheduled	Weight
235-14-7854	Sandra		Smith	09/03/2015	141.5
821-13-2108	Laura		Turner	10/18/2015	167.8

Join 101



MySQL®

Microsoft® SQL Server®

IBM DB2

Microsoft® Access®

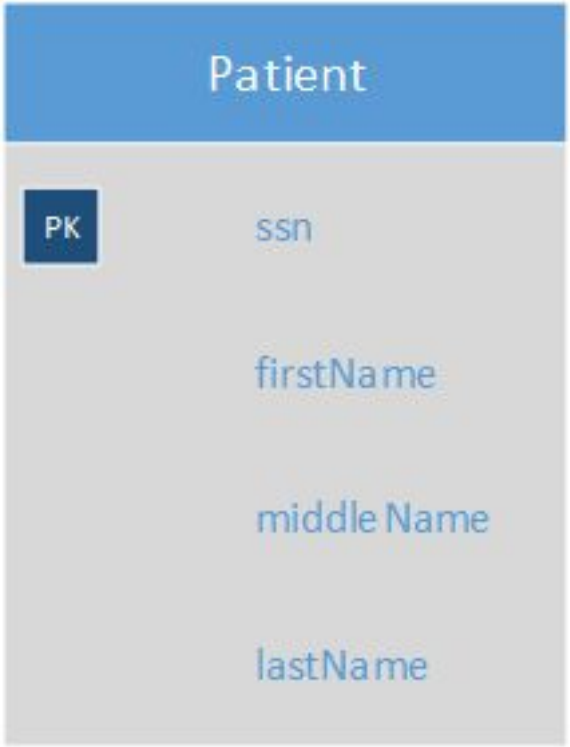
ORACLE® DATABASE

PostgreSQL

Relational model usage

Strong Entity Sets

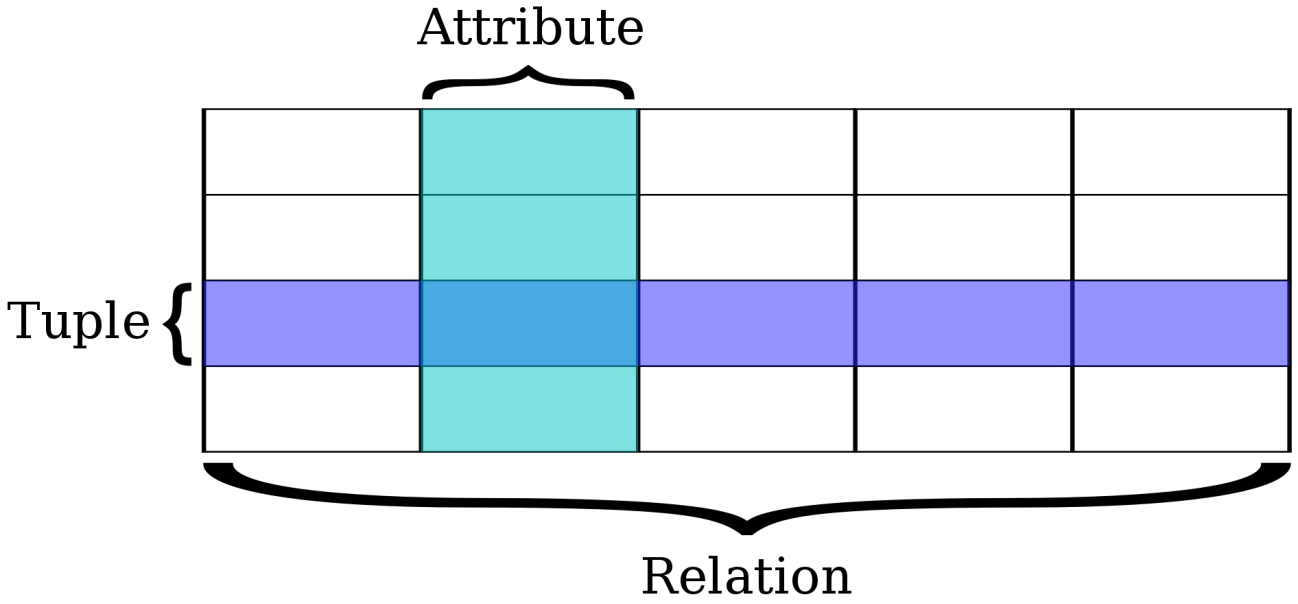
- Attributes of the entity set become attributes of the relation
- Primary keys are maintained



Patient				
ssn	firstName	middleName	lastName	primaryDoctor
235-14-7854	Sandra	null	Smith	943-23-9874
192-48-0924	John	Richard	Moore	862-74-3611

Doctor		
ssn	firstName	lastName
943-23-9874	Rachel	Wang
862-74-3611	Chris	Patel

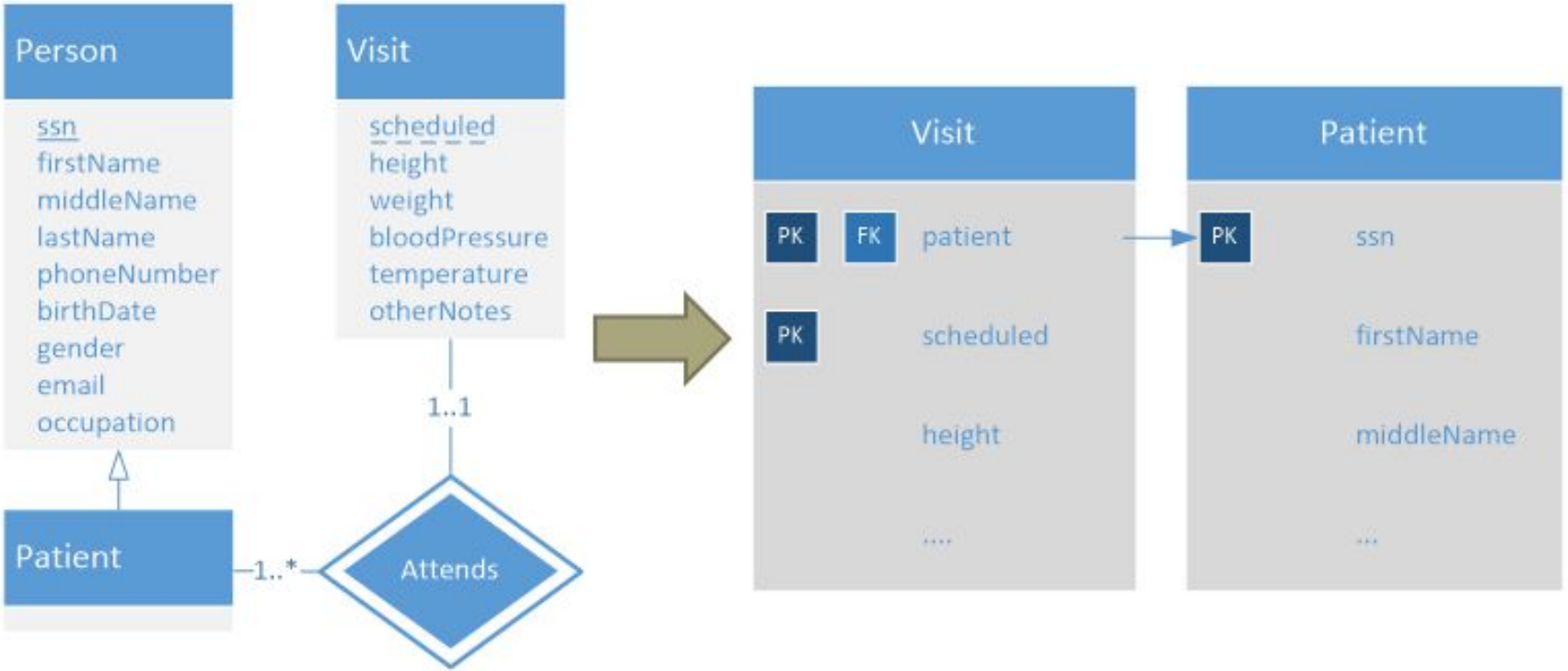
Relation Instances



Types

Types

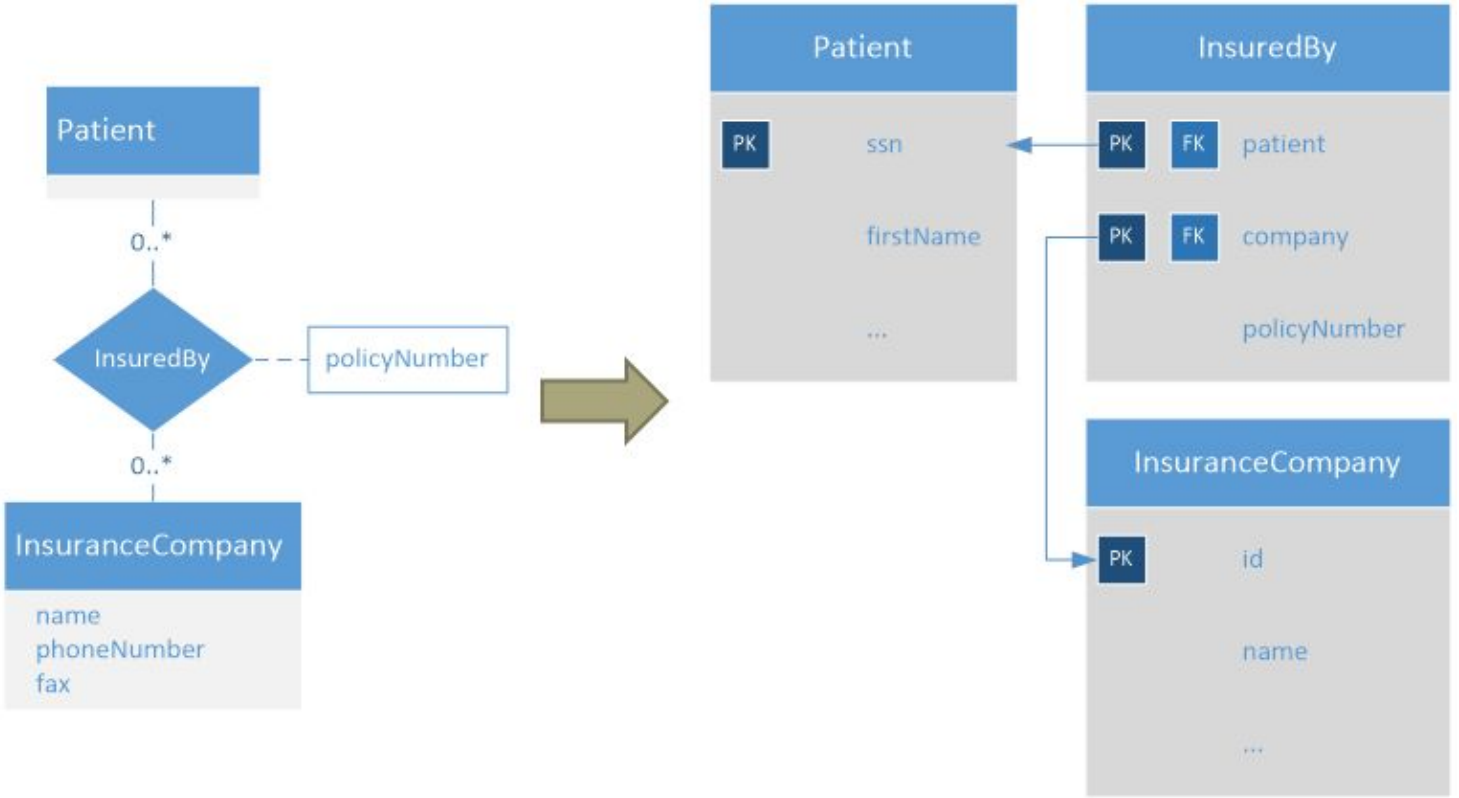
- String
  - Fixed (CHAR)
  - Variable length (VARCHAR, TEXT)
- Numbers
  - Integer (INTEGER)
  - Floating point (DECIMAL)
- Dates
  - Date (DATE)
  - Times (TIME)
  - Date and time (TIMESTAMP)



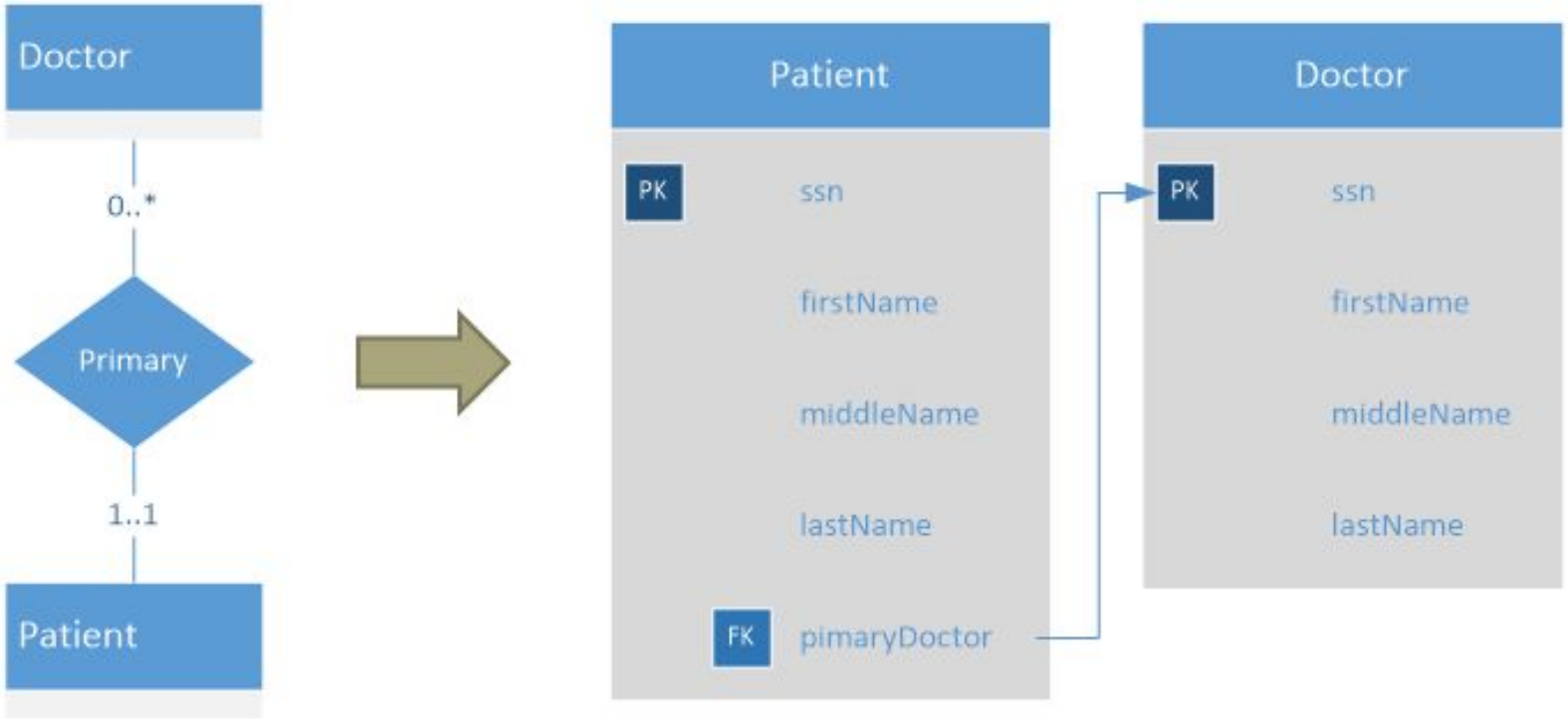
Weak Entity Sets

Weak Relationship Sets

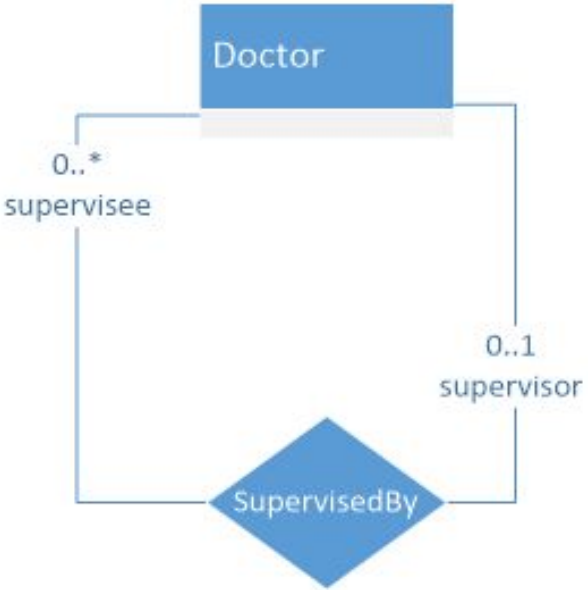
- We can discard these since the relationship is captured by the weak entity set
- Example: The **Attends** relationship is captured by the **Visit** relation created from the weak entity set **Visit**

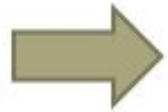


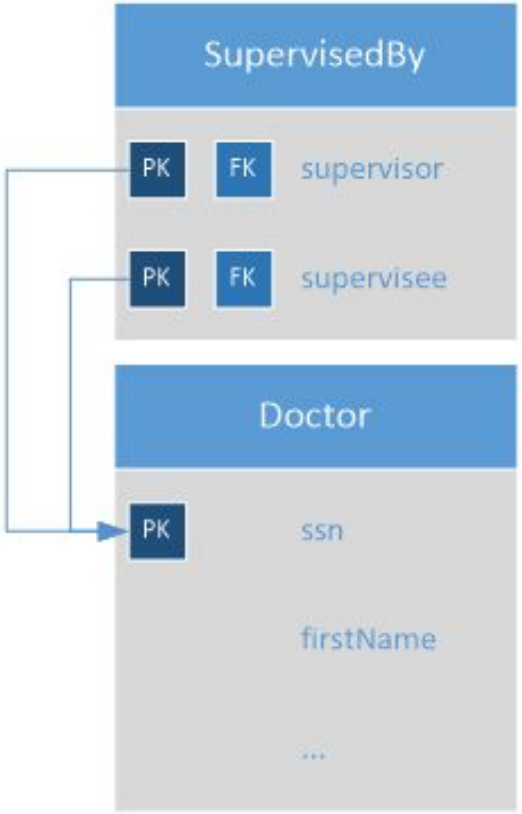
Strong Relationships



Optimization


Doctor
0..*
supervisee
0..1
supervisor
SupervisedBy

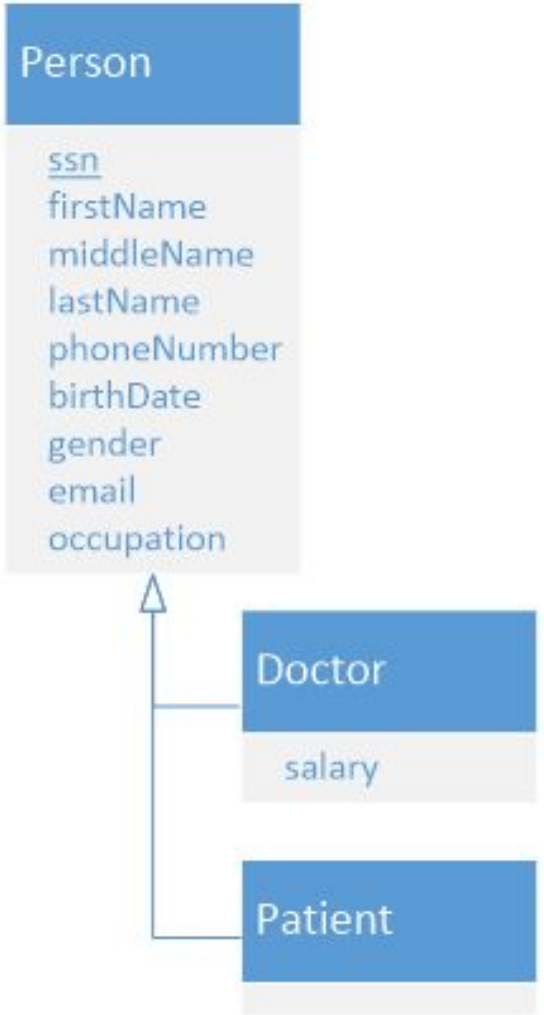


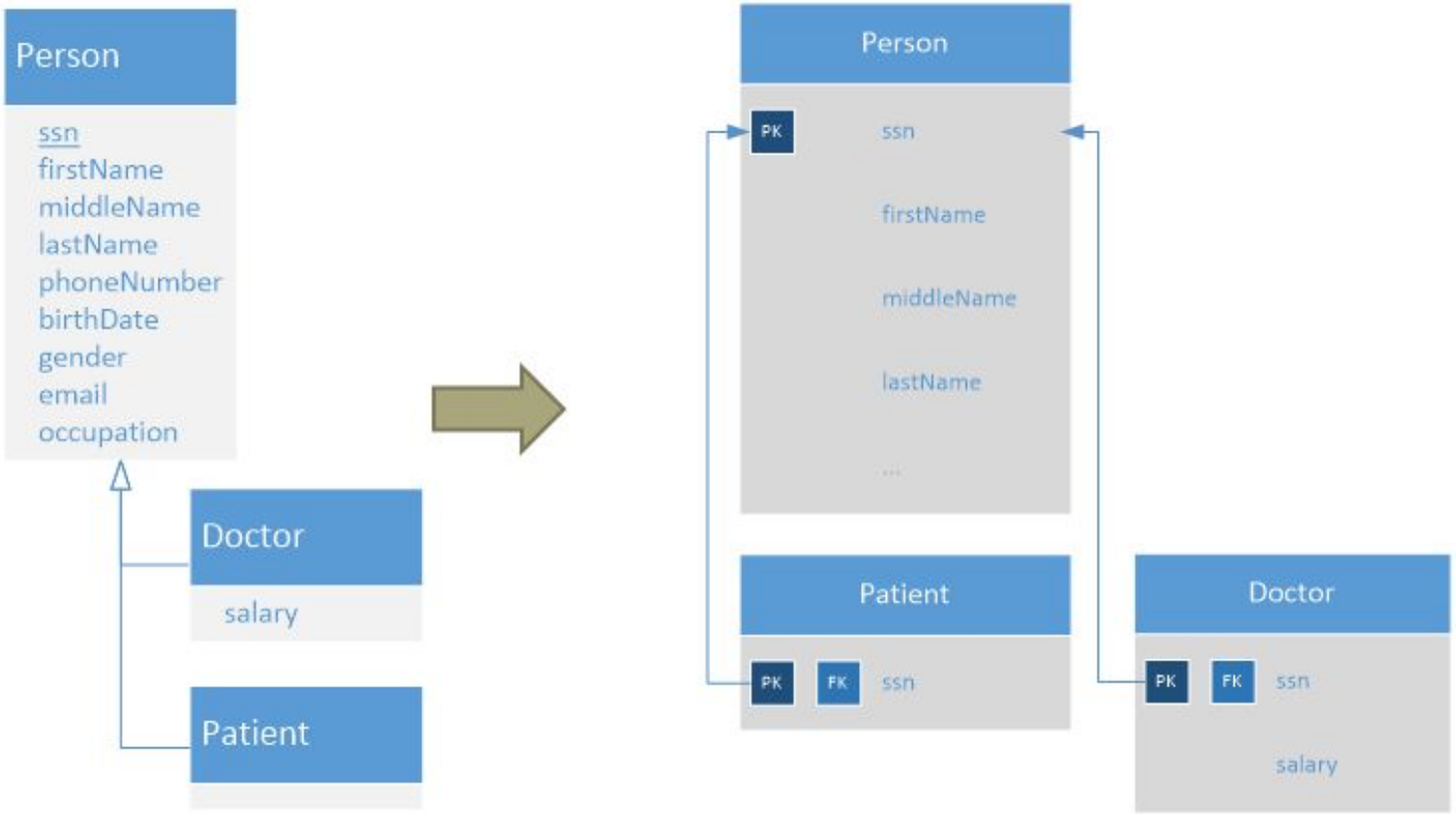

SupervisedBy
PK
FK
supervisor
PK
FK
supervisee
Doctor
PK
ssn
firstName
...

Roles

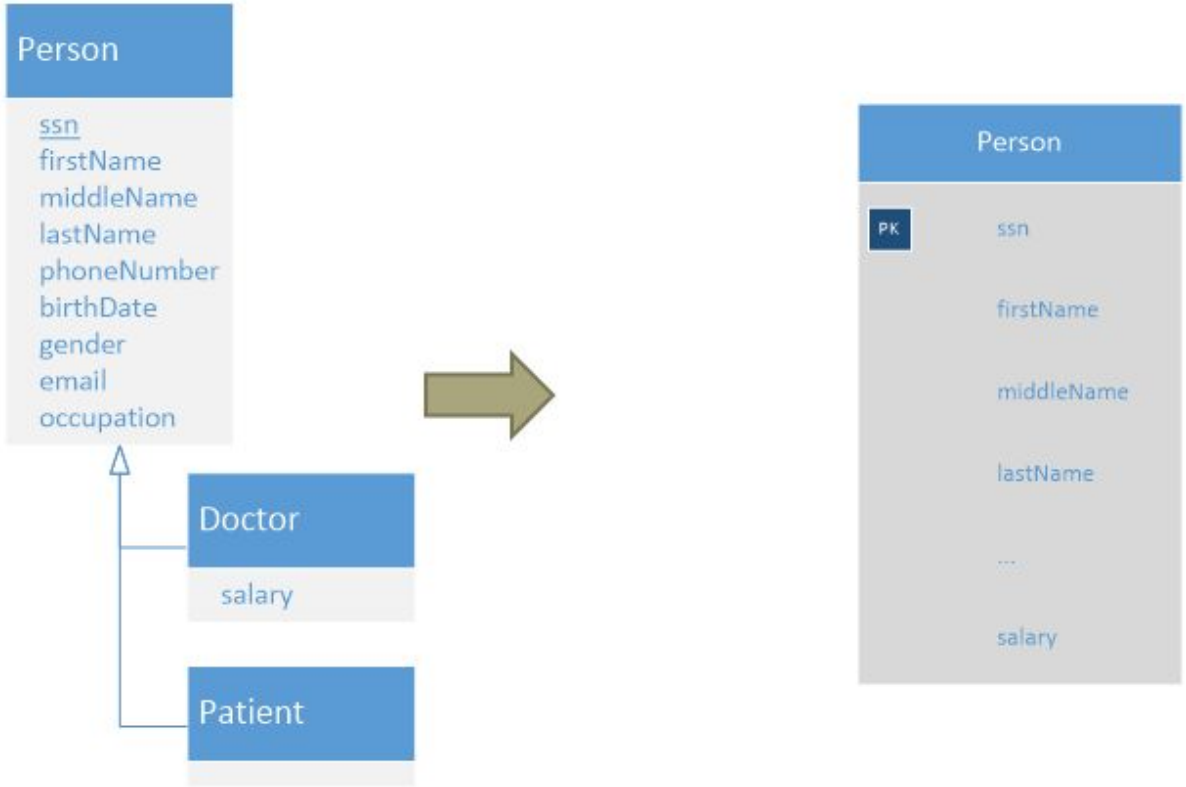
Inheritance Strategies

- Whole
- Top
- Bottom

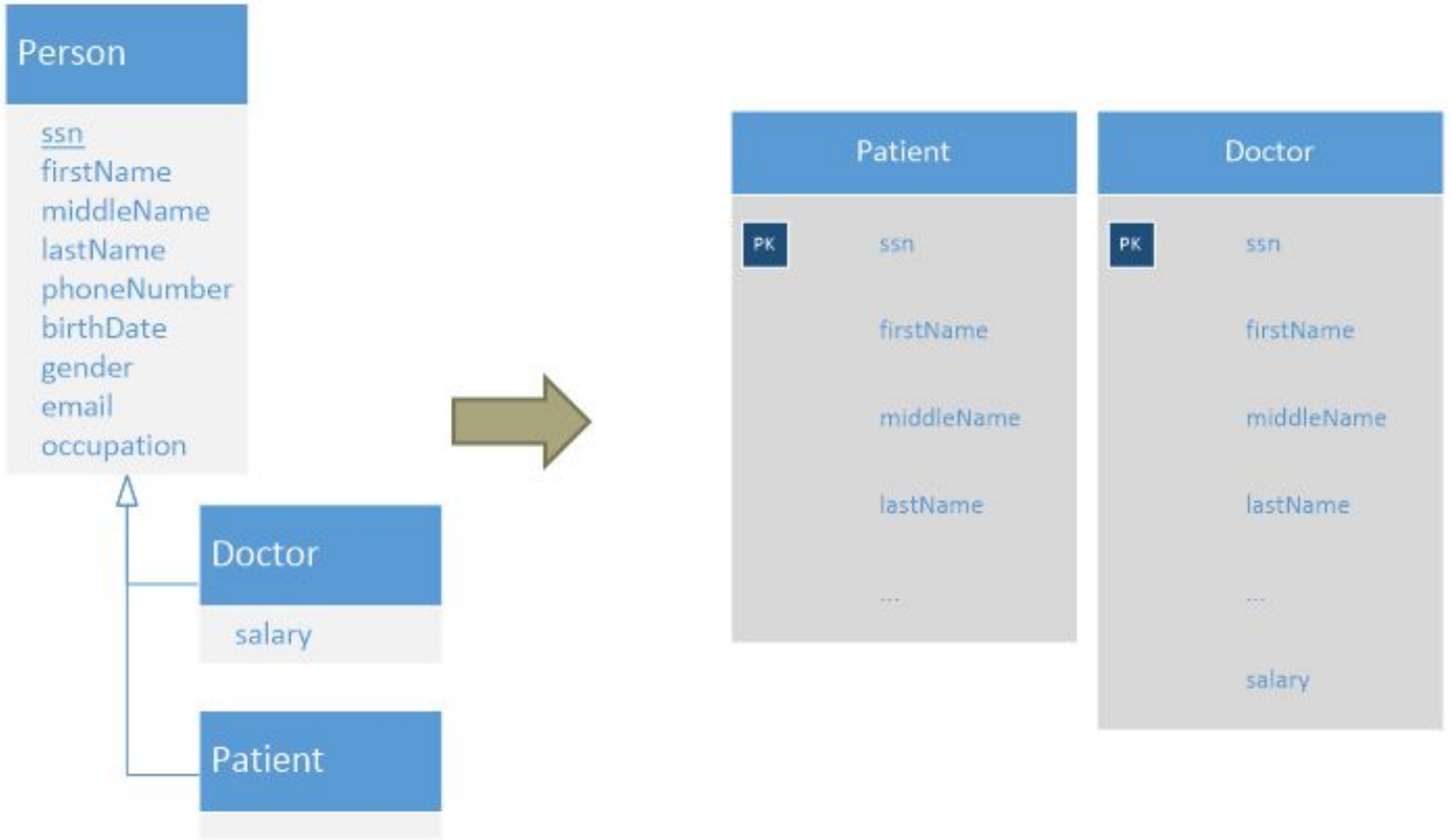




Whole hierarchy



Top of the hierarchy



Bottom of the hierarchy

Roadmap

1. Overview of database engines
2. Data modeling
3. Entity-relationship modeling
4. Relational model
5. Modeling hints

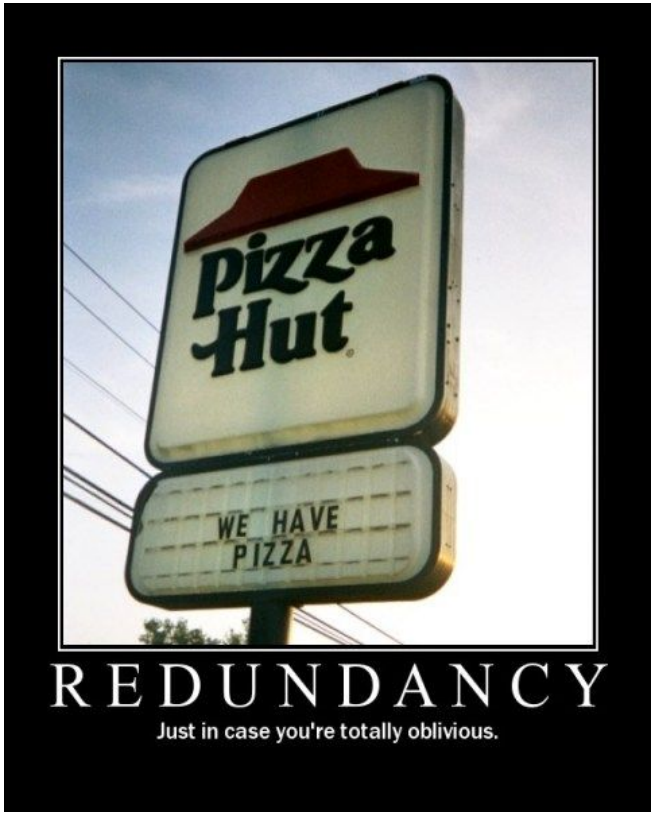
Use your brains!

Database Design

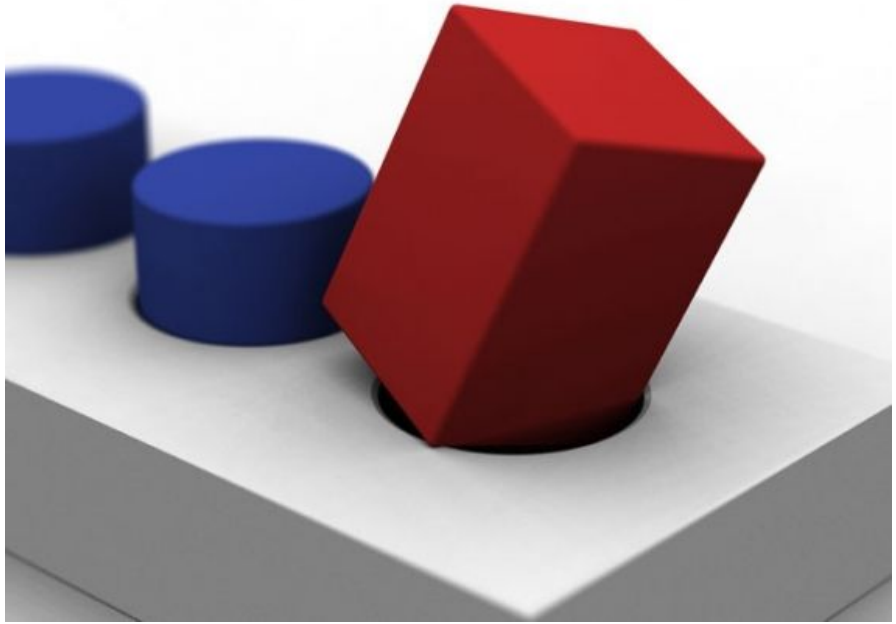
Bad smells in design

Database Design

- Hint #1: redundancy



- Hint #2: incompleteness



- Hint #3: multiple models

Drugs

Patients

Furniture

DDL

- Data Definition Language (DDL)
- Specific notation for defining database schema
- Similar to the Java compiler changing your class into a template

DDL

- Dictionary (metadata - data about data)
  - Database schema (logical)
  - Integrity constraints
    - Primary keys
    - Foreign keys
    - Constraints (e.g. balance >= 0)
    - ...
  - Authorization - who can access what

Creating Tables

```
CREATE TABLE Patient (
    ssn CHARACTER(9),
    firstName VARCHAR(75) NOT NULL,
    middleName VARCHAR(75),
    lastName VARCHAR(75) NOT NULL,
    primaryDoctor CHARACTER(9),
    PRIMARY KEY (ssn),
    FOREIGN KEY (primaryDoctor)
    REFERENCES Doctor(ssn)
);
```