Boyce-Codd Normal Form

- We say a relation *R* is in *BCNF* if whenever $X \rightarrow Y$ is a nontrivial FD that holds in *R*, *X* is a superkey
  - Remember: *nontrivial* means *Y* is not contained in *X*
  - Remember, a *superkey* is any superset of a key (not necessarily a proper superset)

Example

Drinkers(name, addr, beersLiked, manf, favBeer)

FD's: name → addr favBeer, beersLiked → manf

- Only key is {name, beersLiked}
- In each FD, the left side is *not* a superkey
- Any one of these FD's shows *Drinkers* is not in BCNF

Another Example

Beers(name, manf, manfAddr)

FD’s: name → manf, manf → manfAddr

- Only key is {name}
- Name → manf does not violate BCNF, but manf → manfAddr does

Decomposition into BCNF

- Given: relation *R* with FD's *F*
- Look among the given FD's for a BCNF violation $X \rightarrow Y$
  - If any FD following from *F* violates BCNF, then there will surely be an FD in *F* itself that violates BCNF
- Compute X^+
  - Not all attributes, or else X is a superkey

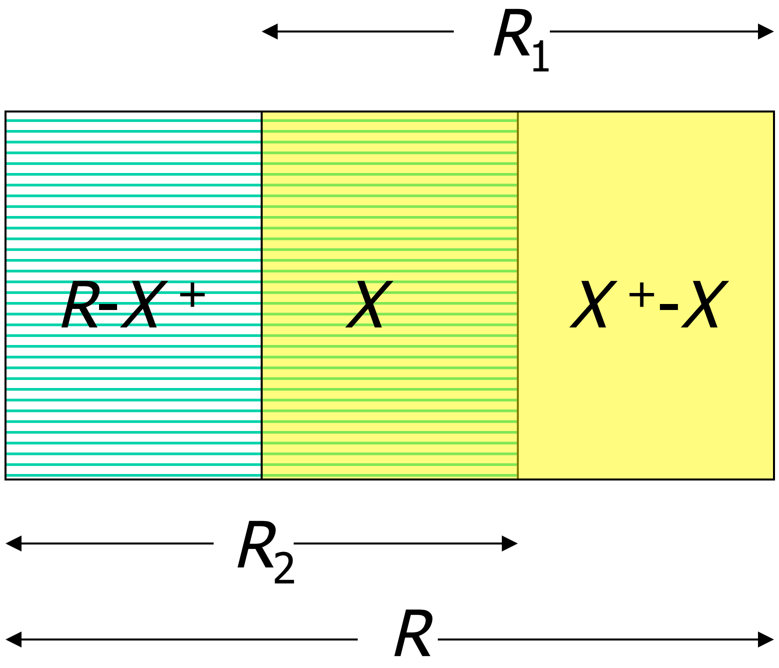
Decompose R Using $X \rightarrow Y$

- Replace R by relations with schemas:

1. $R_1 = X^+$
2. $R_2 = R - (X^+ - X)$

- *Project* given FD's F onto the two new relations

Decomposition Picture



Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)

F = name → addr, name → favBeers
beersLiked → manf

- Pick BCNF violation name → addr
- Close the left side:
  $\{\text{name}\}^+ = \{\text{name, addr, favBeer}\}$
- Decomposed relations:

1. Drinkers1(name, addr, favBeer)
2. Drinkers2(name, beersLiked, manf)

Example: BCNF Decomposition

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF
- Projecting FD's is easy here
- For Drinkers1(name, addr, favBeer), relevant FD's are name → addr and name → favBeer
  - Thus, {name} is the only key and Drinkers1 is in BCNF

Example: BCNF Decomposition

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked → manf, and the only key is {name, beersLiked}
    - Violation of BCNF
- beersLiked$^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:

1. Drinkers3(beersLiked, manf)
2. Drinkers4(name, beersLiked)

Example: BCNF Decomposition

- The resulting decomposition of *Drinkers:*
  1. Drinkers1(name, addr, favBeer)
  2. Drinkers3(beersLiked, manf)
  3. Drinkers4(name, beersLiked)
  - Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like
  - Compare with running example:
  1. Drinkers(name, addr, phone)
  2. Beers(name, manf)
  3. Likes(drinker,beer)

Third Normal Form – Motivation

- There is one structure of FD's that causes trouble when we decompose
- $AB \rightarrow C$ and $C \rightarrow B$
  - Example:
    A = street address, B = city, C = post code
- There are two keys, $\{A,B\}$ and $\{A,C\}$
- $C \rightarrow B$ is a BCNF violation, so we must decompose into AC, BC

We Cannot Enforce FD’s

- The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD $AB \rightarrow C$ by checking FD’s in these decomposed relations
- Example with *A* = street, *B* = city, and *C* = post code on the next slide

An Unenforceable FD

street	post
Campusvej	5230
Vestergade	5000

city	post
Odense	5230
Odense	5000

Join tuples with equal post codes

street	city	post
Campusvej	Odense	5230
Vestergade	Odense	5000

No FD's were violated in the decomposed relations and FD street city → post holds for the database as a whole

An Unenforceable FD

street	post
Hjallesevej	5230
Hjallesevej	5000

city	post
Odense	5230
Odense	5000

Join tuples with equal post codes

street	city	post
Hjallesevej	Odense	5230
Hjallesevej	Odense	5000

Although no FD’s were violated in the decomposed relations, FD street city → post is violated by the database as a whole

Another Unenforcable FD

- Departures(time, track, train)
- time track → train and train → track
- Two keys, {time,track} and {time,train}
- train → track is a BCNF violation, so we must decompose into
  Departures1(time, train)
  Departures2(track,train)

Another Unenforceable FD

time	train
19:08	ICL54
19:16	IC852

tracktrain	
4	ICL54
3	IC852

Join tuples with equal train code

time	track	train
19:08	4	ICL54
19:16	3	IC852

No FD’s were violated in the decomposed relations,
FD time track → train holds for the database as a whole

Another Unenforceable FD

time	train
19:08	ICL54
19:08	IC 42

tracktrain	
4	ICL54
4	IC 42

Join tuples with equal train code

time	track	train
19:08	4	ICL54
19:08	4	IC 42

Although no FD’s were violated in the decomposed relations,
FD time track → train is violated by the database as a whole

3NF Let’ s Us Avoid This Problem

- 3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation
- An attribute is *prime* if it is a member of any key
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime

Example: 3NF

- In our problem situation with FD's $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC
- Thus A, B, and C are each prime
- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF

What 3NF and BCNF Give You

- There are two important properties of a decomposition:

1. *Lossless Join:* it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original
2. *Dependency Preservation:* it should be possible to check in the projected relations whether all the given FD's are satisfied

3NF and BCNF – Continued

- We can get (1) with a BCNF decomposition
- We can get both (1) and (2) with a 3NF decomposition
- But we can’t always get (1) and (2) with a BCNF decomposition
  - street-city-post is an example
  - time-track-train is another example

Testing for a Lossless Join

- If we project R onto R_1, R_2,…, R_k , can we recover R by rejoining?
- Any tuple in R can be recovered from its projected fragments
- So the only question is: when we rejoin, do we ever get back something we didn't have originally?

The Chase Test

- Suppose tuple *t* comes back in the join
- Then *t* is the join of projections of some tuples of *R*, one for each R_i of the decomposition
- Can we use the given FD's to show that one of these tuples must be *t* ?

The Chase – (2)

- Start by assuming $t = abc\ldots$.
- For each i, there is a tuple s_i of R that has a, b, c,… in the attributes of R_i
- s_i can have any values in other attributes
- We'll use the same letter as in t, but with a subscript, for these components

Example: The Chase

- Let R = $ABCD$, and the decomposition be AB, BC, and CD
- Let the given FD's be $C \rightarrow D$ and $B \rightarrow A$
- Suppose the tuple t = abcd is the join of tuples projected onto AB, BC, CD

The *Tableau*

The tuples of R projected onto AB, BC, CD

A	B	C	D
a	b	c_1	d_1
$\cancel{a_2}\, a$	b	c	$\cancel{d_2}\, d$
a_3	b_3	c	d

Use $B \rightarrow A$

Use $C \rightarrow D$

We've proved the second tuple must be t

Summary of the Chase

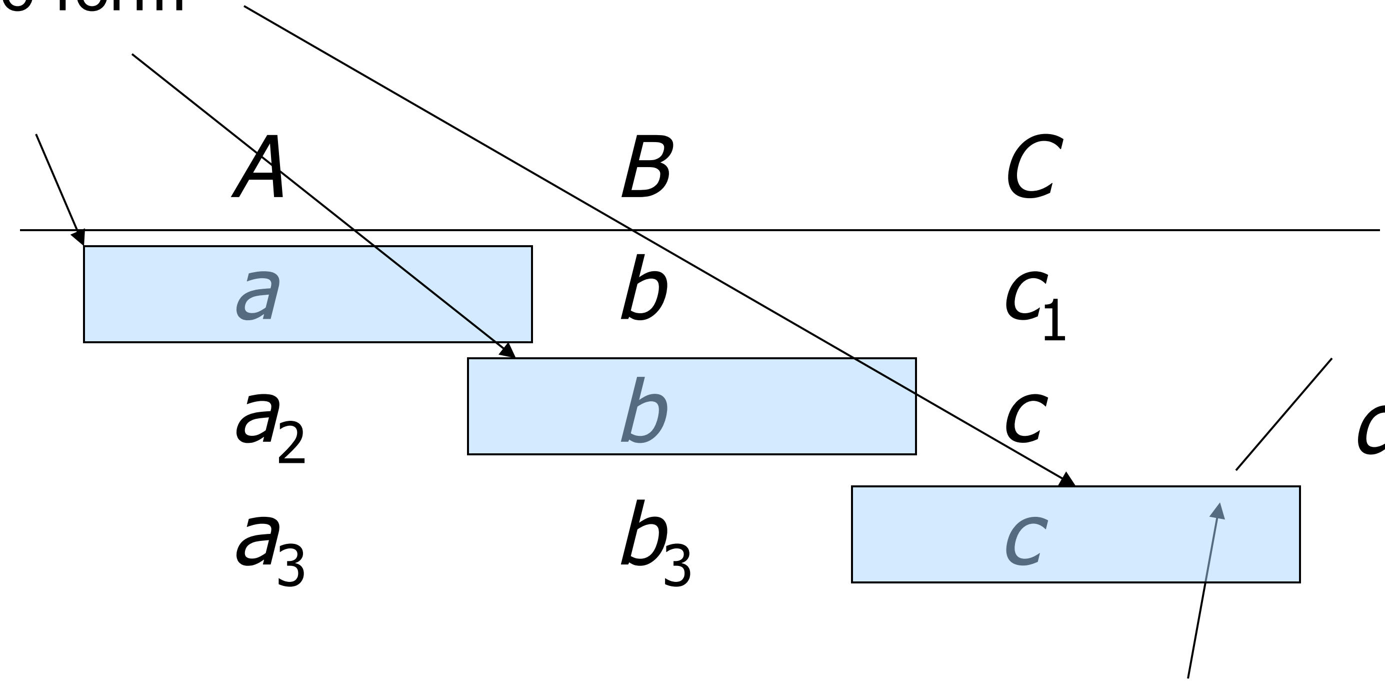
1. If two rows agree in the left side of a FD, make their right sides agree too
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless)
4. Otherwise, the final tableau is a counterexample

Example: Lossy Join

- Same relation $R = ABCD$ and same decomposition.
- But with only the FD $C \to D$

The *Tableau*

These projections rejoin to form *abcd*



A	B	C	D
a	b	c_1	d_1
a_2	b	c	$d\,d_2$
a_3	b_3	c	d

These three tuples are an example *R* that shows the join lossy
abcd is not in *R*, but we can project and rejoin to get *abcd*

Use $C \rightarrow D$

3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation
- Need *minimal basis* for the FD's:
  1. Right sides are single attributes
  2. No FD can be removed
  3. No attribute can be removed from a left side

Constructing a Minimal Basis

1. Split right sides
2. Repeatedly try to remove an FD and see if the remaining FD’s are equivalent to the original
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD’s are equivalent to the original

3NF Synthesis – (2)

- One relation for each FD in the minimal basis
  - Schema is the union of the left and right sides
- If no key is contained in an FD, then add one relation whose schema is some key

Example: 3NF Synthesis

- Relation R = ABCD
- FD's $A \to B$ and $A \to C$
- Decomposition: AB and AC from the FD's, plus AD for a key

Why It Works

- Preserves dependencies: each FD from a minimal basis is contained in a relation, thus preserved
- Lossless Join: use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables
- 3NF: hard part – a property of minimal bases

Summary 5

More things you should know:

- Functional Dependency
- Key, Superkey
- Update Anomaly, Deletion Anomaly
- BCNF, Closure, Decomposition
- Chase Algorithm
- 3rd Normal Form

Entity-Relationship Model

Purpose of E/R Model

- The E/R model allows us to sketch database schema designs
  - Includes some constraints, but not operations
- Designs are pictures called *entity-relationship diagrams*
- Later: convert E/R designs to relational DB designs

Framework for E/R

- Design is a serious business
- The “boss” knows they want a database, but they don’t know what they want in it
- Sketching the key components is an efficient way to develop a working database

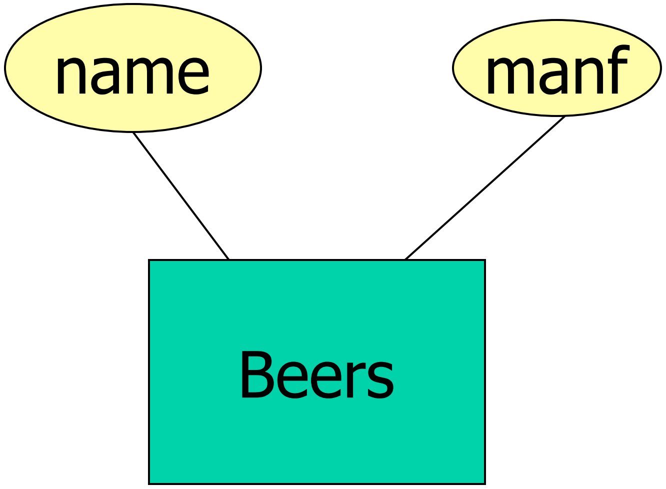
Entity Sets

- *Entity* = "thing" or object
- *Entity set* = collection of similar entities
  - Similar to a class in object-oriented languages
- *Attribute* = property of (the entities of) an entity set
  - Attributes are simple values, e.g. integers or character strings, not structs, sets, etc.

E/R Diagrams

- In an entity-relationship diagram:
  - Entity set = rectangle
  - Attribute = oval, with a line to the rectangle representing its entity set

Example:

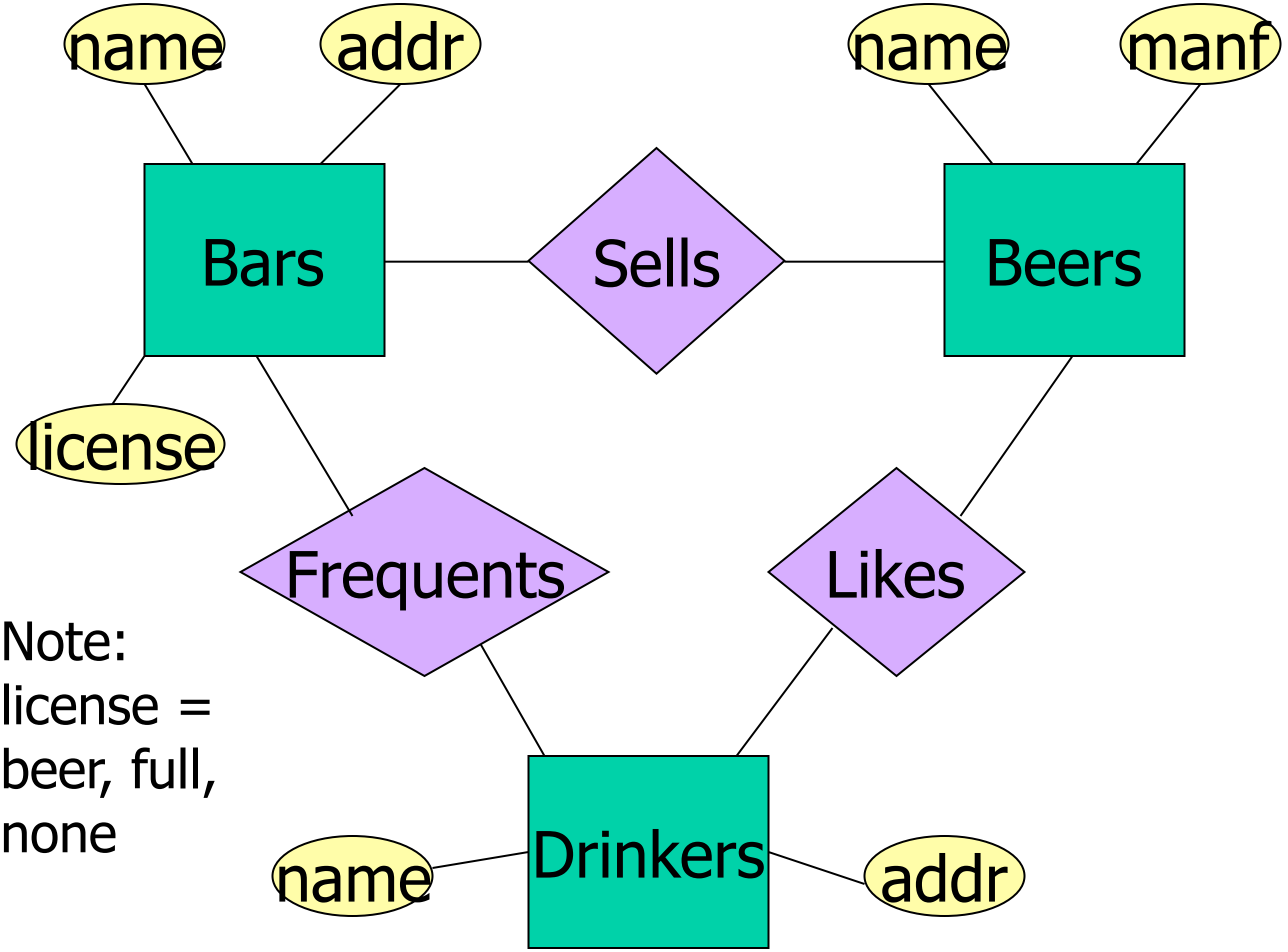


- Entity set Beers has two attributes, name and manf (manufacturer)
- Each Beers entity has values for these two attributes, e.g. (Odense Classic, Albani)

Relationships

- A relationship connects two or more entity sets
- It is represented by a diamond, with lines to each of the entity sets involved

Example: Relationships



Note:
license =
beer, full,
none

Bars sell some beers

Drinkers like some beers

Drinkers frequent some bars

Relationship Set

- The current “value” of an entity set is the set of entities that belong to it
  - Example: the set of all bars in our database
- The “value” of a relationship is a *relationship set*, a set of tuples with one component for each related entity set

Example: Relationship Set

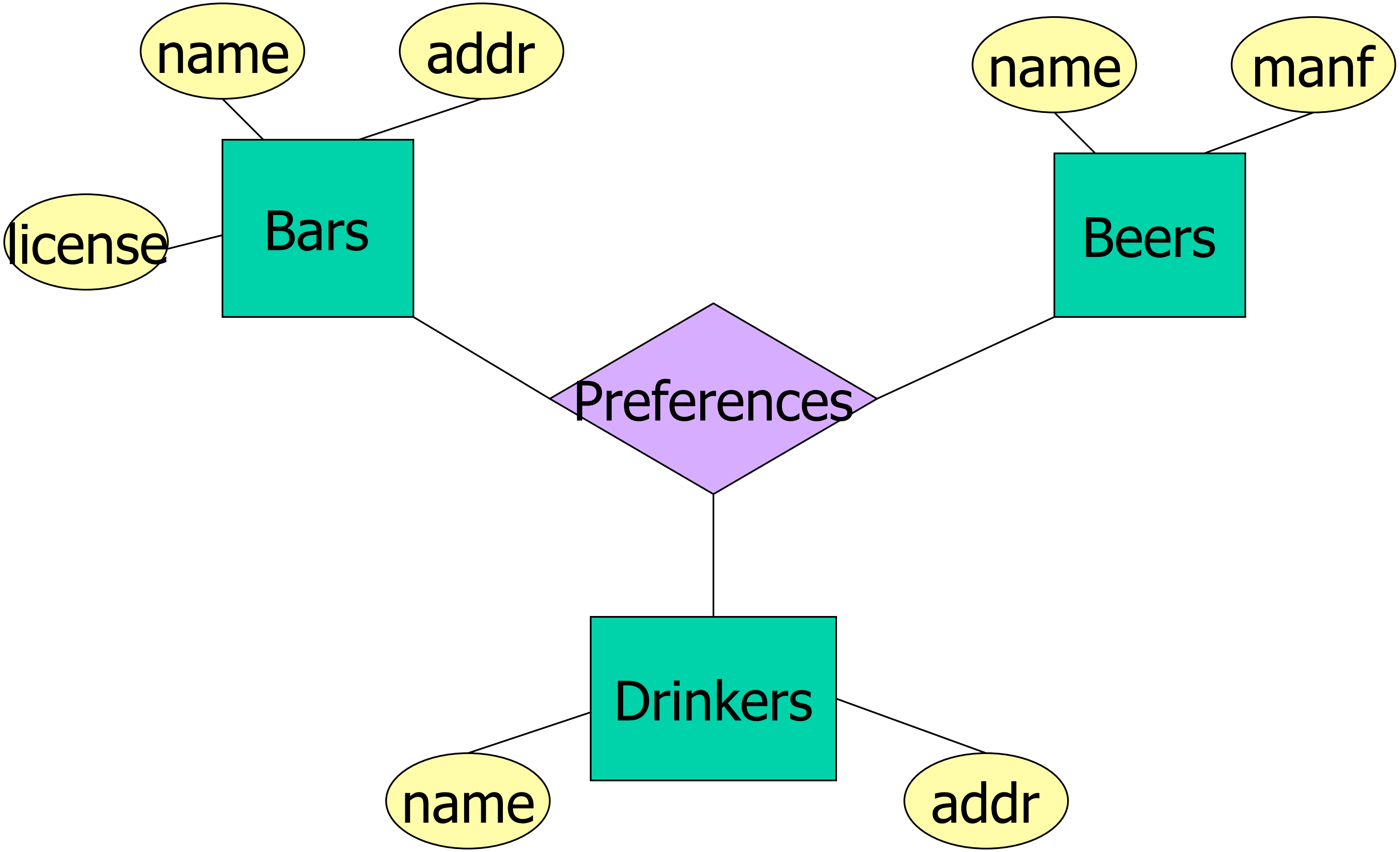
- For the relationship Sells, we might have a relationship set like:

Bar	Beer
C.Ch.	Od.Cl.
C.Ch.	Erd.Wei.
C.Bio.	Od.Cl.
Brygg.	Pilsener
C4	Erd.Wei.

Multiway Relationships

- Sometimes, we need a relationship that connects more than two entity sets
- Suppose that drinkers will only drink certain beers at certain bars
  - Our three binary relationships Likes, Sells, and Frequents do not allow us to make this distinction
  - But a 3-way relationship would

Example: 3-Way Relationship



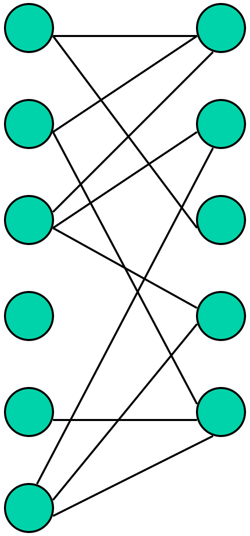
A Typical Relationship Set

Bar	Drinker	Beer
C.Ch.	Peter	Erd.Wei.
C.Ch.	Lars	Od.Cl.
C.Bio.	Peter	Od.Cl.
Brygg.	Peter	Pilsener
C4	Peter	Erd.Wei.
C.Bio.	Lars	Tuborg
Brygg.	Lars	Ale

Many-Many Relationships

- Focus: binary relationships, such as Sells between Bars and Beers
- In a *many-many* relationship, an entity of either set can be connected to many entities of the other set
  - E.g., a bar sells many beers; a beer is sold by many bars

In Pictures:

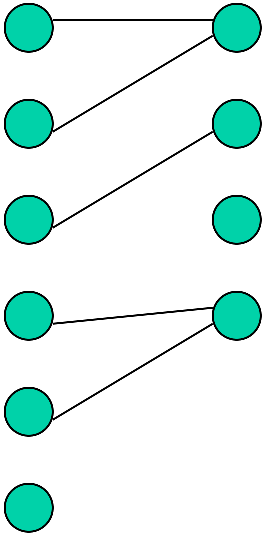


many-many

Many-One Relationships

- Some binary relationships are *many - one* from one entity set to another
- Each entity of the first set is connected to at most one entity of the second set
- But an entity of the second set can be connected to zero, one, or many entities of the first set

In Pictures:



many-one

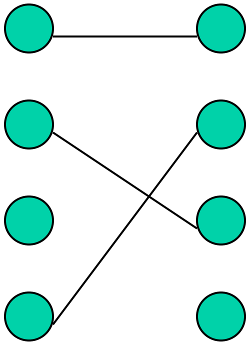
Example: Many-One Relationship

- Favorite, from Drinkers to Beers is many-one
- A drinker has at most one favorite beer
- But a beer can be the favorite of any number of drinkers, including zero

One-One Relationships

- In a *one-one* relationship, each entity of either entity set is related to at most one entity of the other set
- Example: Relationship Best-seller between entity sets Manfs (manufacturer) and Beers
    - A beer cannot be made by more than one manufacturer, and no manufacturer can have more than one best-seller (assume no ties)

In Pictures:

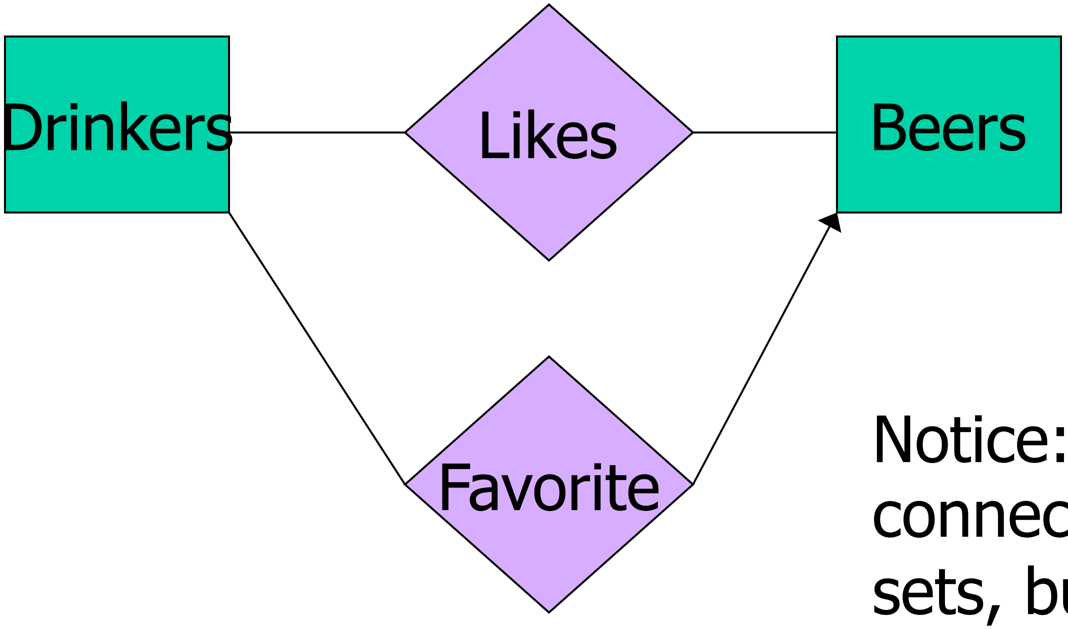


one-one

Representing “Multiplicity”

- Show a many-one relationship by an arrow entering the “one” side
  - Remember: Like a functional dependency
- Show a one-one relationship by arrows entering both entity sets
- Rounded arrow = “exactly one,” i.e., each entity of the first set is related to exactly one entity of the target set

Example: Many-One Relationship

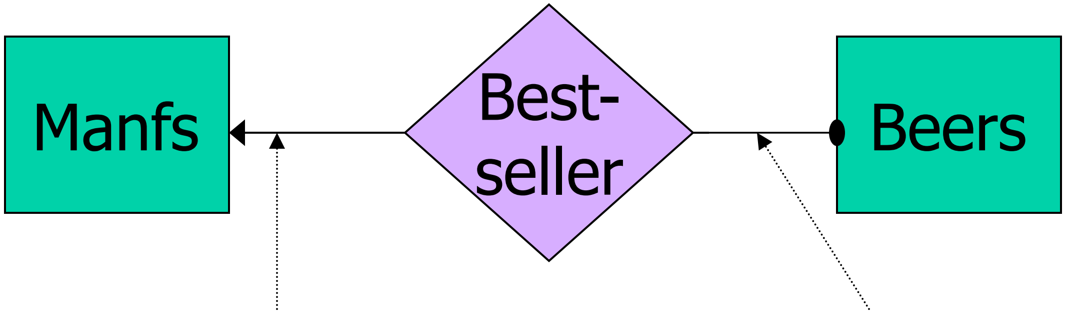


Notice: two relationships connect the same entity sets, but are different

Example: One-One Relationship

- Consider Best-seller between Manfs and Beers
- Some beers are not the best-seller of any manufacturer, so a rounded arrow to Manfs would be inappropriate.
- But a beer manufacturer has to have a best-seller

In the E/R Diagram



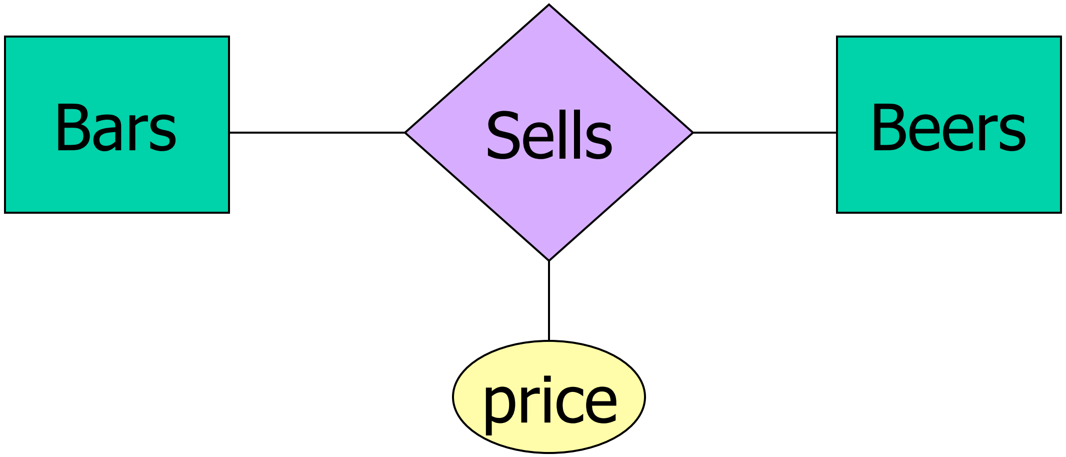
A beer is the best-seller for 0 or 1 manufacturer(s)

A manufacturer has exactly one best seller

Attributes on Relationships

- Sometimes it is useful to attach an attribute to a relationship
- Think of this attribute as a property of tuples in the relationship set

Example: Attribute on Relationship

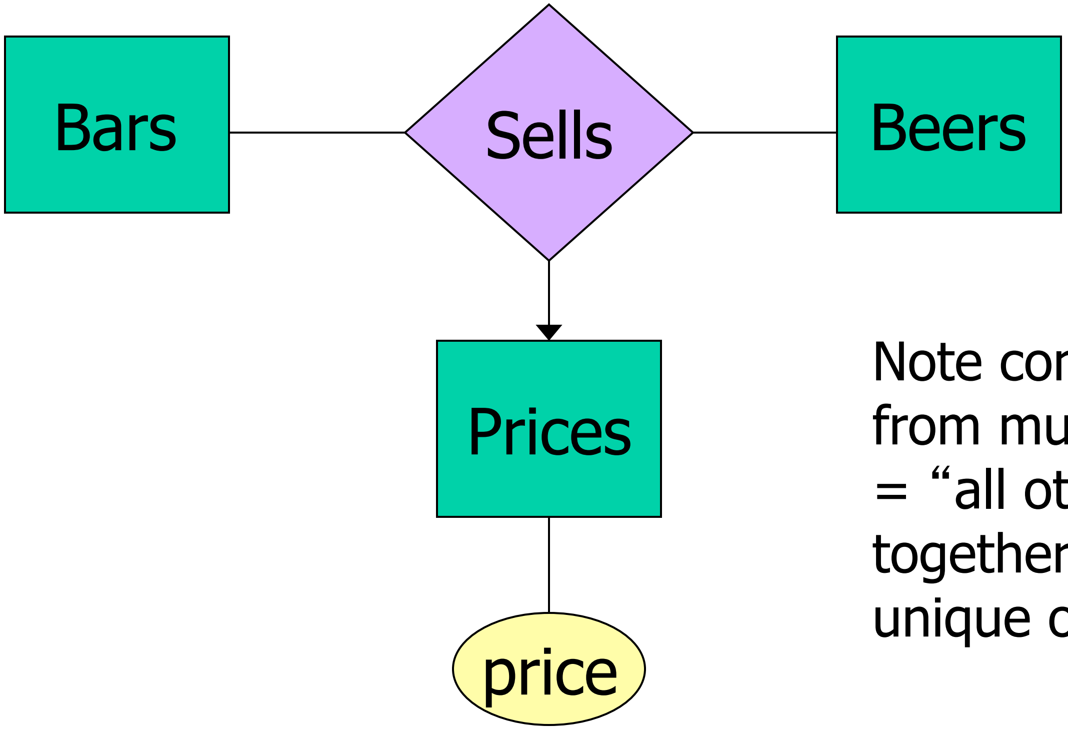


Price is a function of both the bar and the beer, not of one alone

Equivalent Diagrams Without Attributes on Relationships

- Create an entity set representing values of the attribute
- Make that entity set participate in the relationship

Example: Removing an Attribute from a Relationship

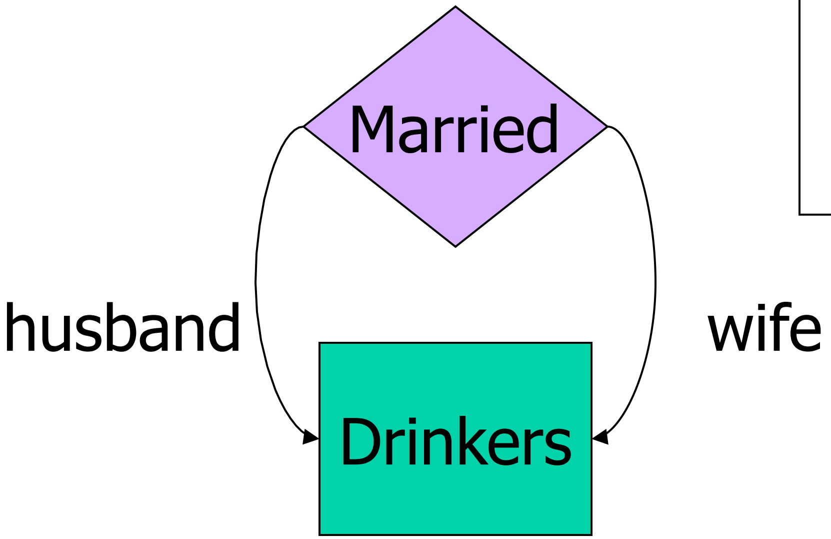


Note convention: arrow from multiway relationship = “all other entity sets together determine a unique one of these”

Roles

- Sometimes an entity set appears more than once in a relationship
- Label the edges between the relationship and the entity set with names called *roles*

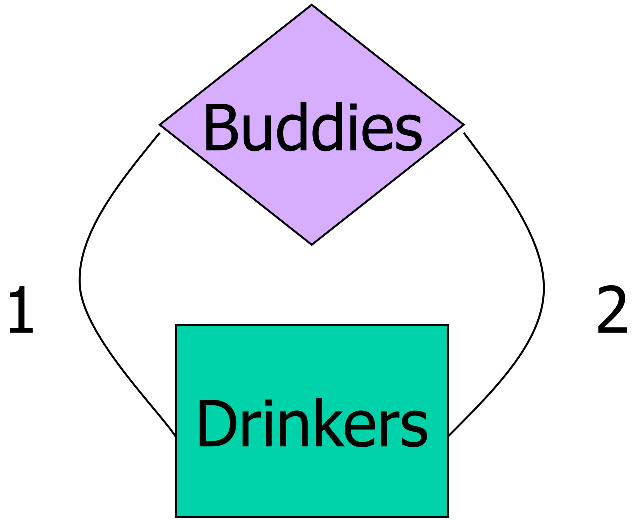
Example: Roles



Relationship Set

Husband	Wife
Lars	Lene
Kim	Joan
…	…

Example: Roles



Relationship Set

Buddy1	Buddy2
Peter	Lars
Peter	Pepe
Pepe	Bea
Bea	Rafa
…	…

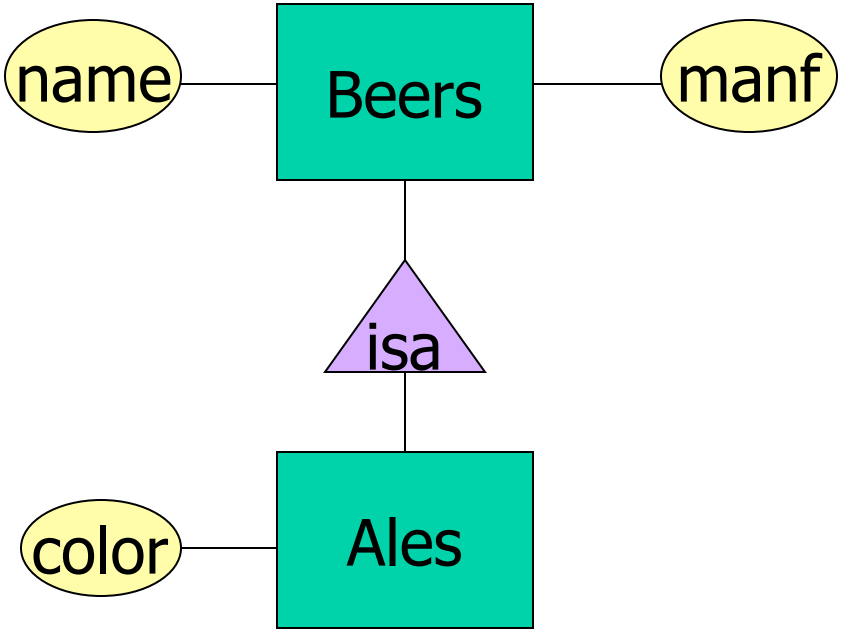
Subclasses

- *Subclass* = special case = fewer entities = more properties
- Example: Ales are a kind of beer
  - Not every beer is an ale, but some are
  - Let us suppose that in addition to all the *properties* (attributes and relationships) of beers, ales also have the attribute color

Subclasses in E/R Diagrams

- Assume subclasses form a tree
  - I.e., no multiple inheritance
- Isa triangles indicate the subclass relationship
  - Point to the superclass

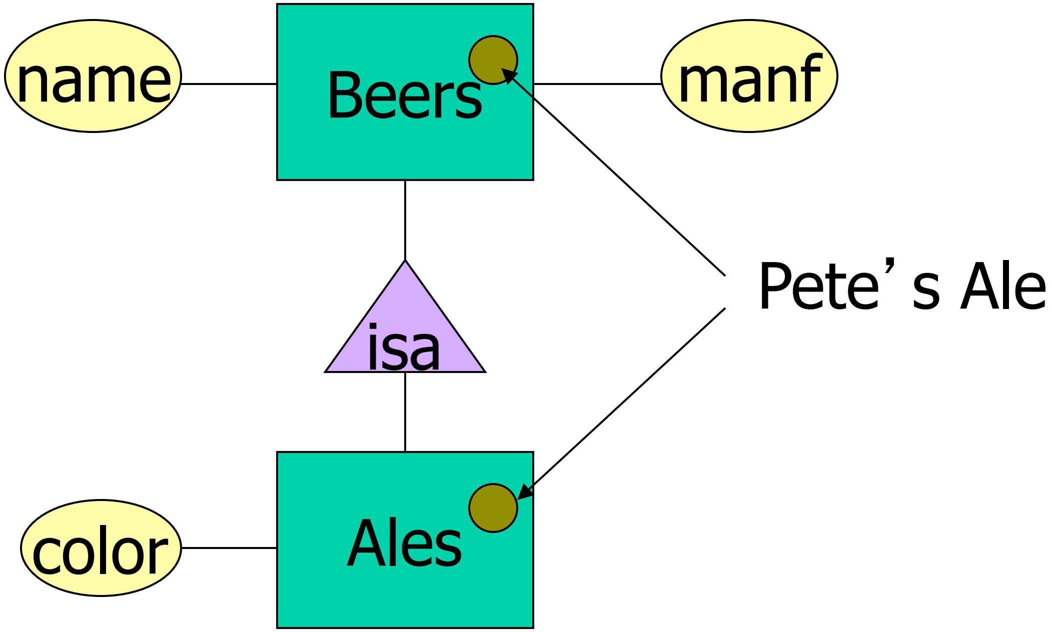
Example: Subclasses



E/R Vs. Object-Oriented Subclasses

- In OO, objects are in one class only
  - Subclasses inherit from superclasses.
- In contrast, E/R entities have *representatives* in all subclasses to which they belong
  - Rule: if entity *e* is represented in a subclass, then *e* is represented in the superclass (and recursively up the tree)

Example: Representatives of Entities



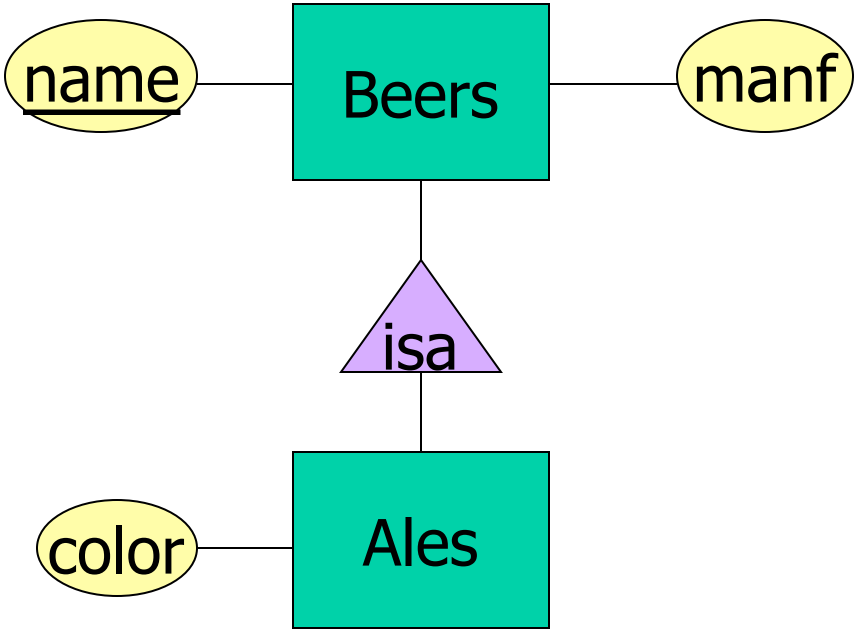
Keys

- A *key* is a set of attributes for one entity set such that no two entities in this set agree on all the attributes of the key
  - It is allowed for two entities to agree on some, but not all, of the key attributes
- We must designate a key for every entity set

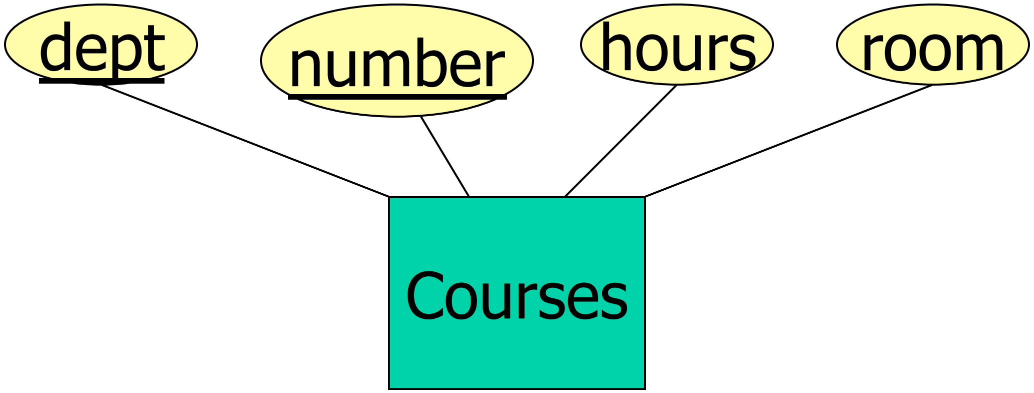
Keys in E/R Diagrams

- Underline the key attribute(s)
- In an Isa hierarchy, only the root entity set has a key, and it must serve as the key for all entities in the hierarchy

Example: name is Key for Beers



Example: a Multi-attribute Key



- Note that hours and room could also serve as a key, but we must select only one key

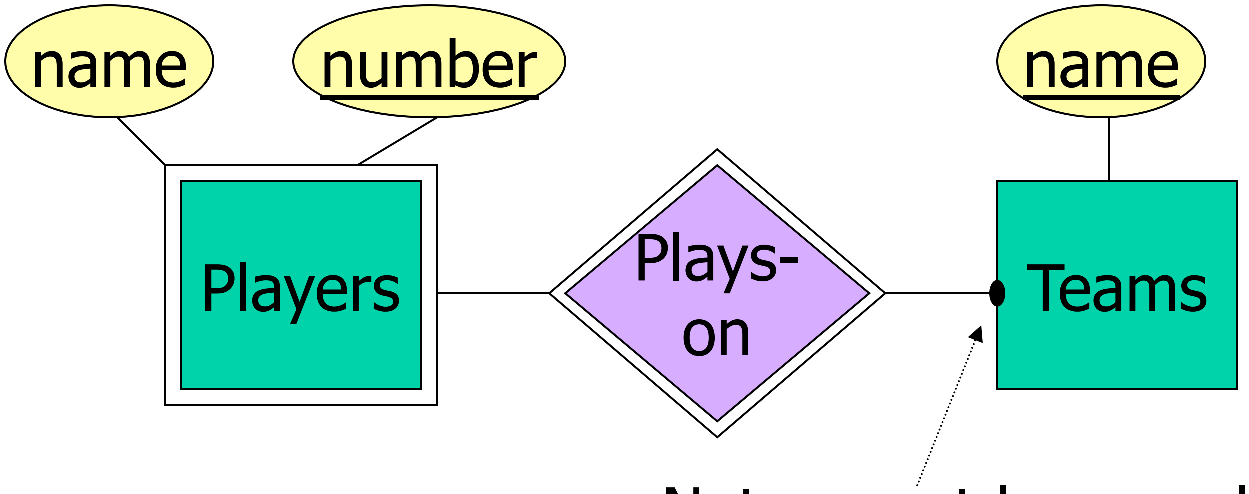
Weak Entity Sets

- Occasionally, entities of an entity set need “help” to identify them uniquely
- Entity set *E* is said to be *weak* if in order to identify entities of *E* uniquely, we need to follow one or more many-one relationships from *E* and include the key of the related entities from the connected entity sets

Example: Weak Entity Set

- name is almost a key for football players, but there might be two with the same name
- number is certainly not a key, since players on two teams could have the same number.
- But number, together with the team name related to the player by Plays-on should be unique

In E/R Diagrams



Note: must be rounded because each player needs a team to help with the key

- Double diamond for *supporting* many-one relationship
- Double rectangle for the weak entity set

Weak Entity-Set Rules

- A weak entity set has one or more many-one relationships to other (supporting) entity sets
  - Not every many-one relationship from a weak entity set need be supporting
  - But supporting relationships must have a rounded arrow (entity at the “one” end is guaranteed)

Weak Entity-Set Rules – (2)

- The key for a weak entity set is its own underlined attributes and the keys for the supporting entity sets
    - E.g., (player) number and (team) name is a key for Players in the previous example

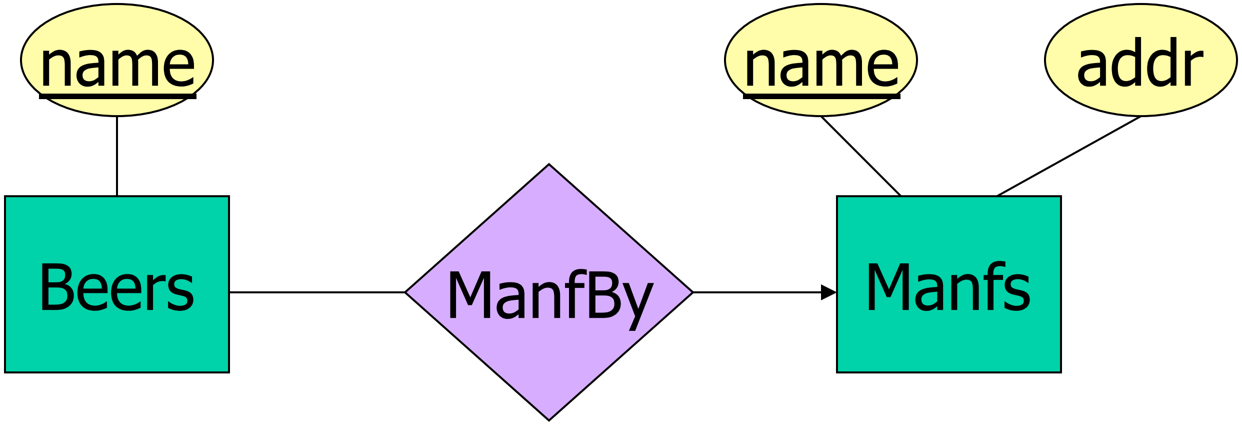
Design Techniques

1. Avoid redundancy
2. Limit the use of weak entity sets
3. Don’t use an entity set when an attribute will do

Avoiding Redundancy

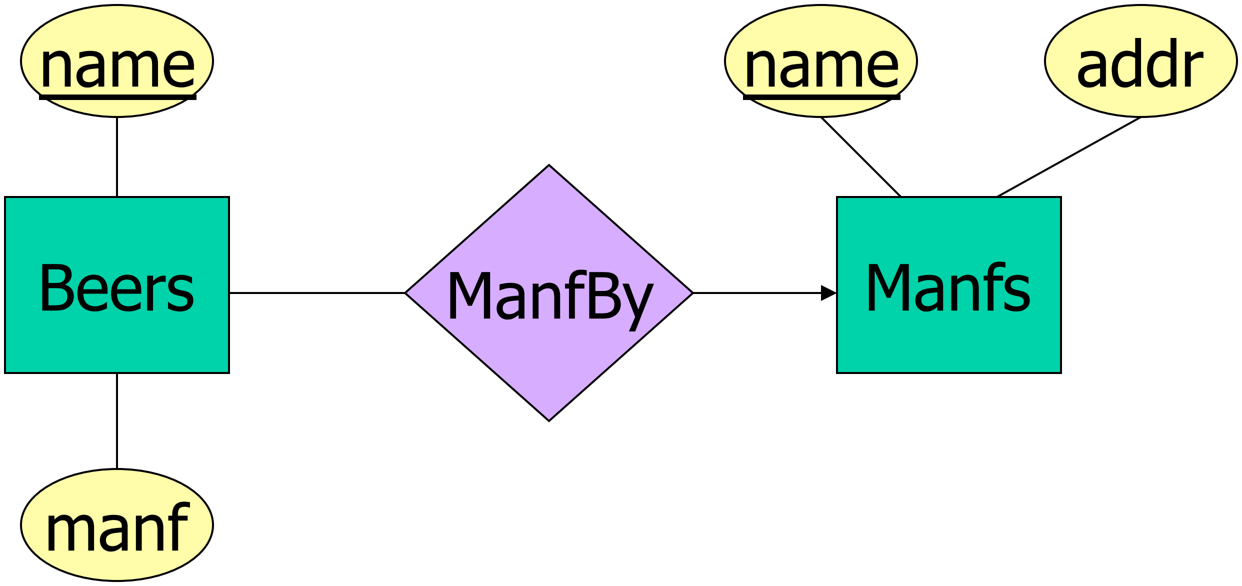
- *Redundancy* = saying the same thing in two (or more) different ways
- Wastes space and (more importantly) encourages inconsistency
    - Two representations of the same fact become inconsistent if we change one and forget to change the other
    - Recall anomalies due to FD's

Example: Good



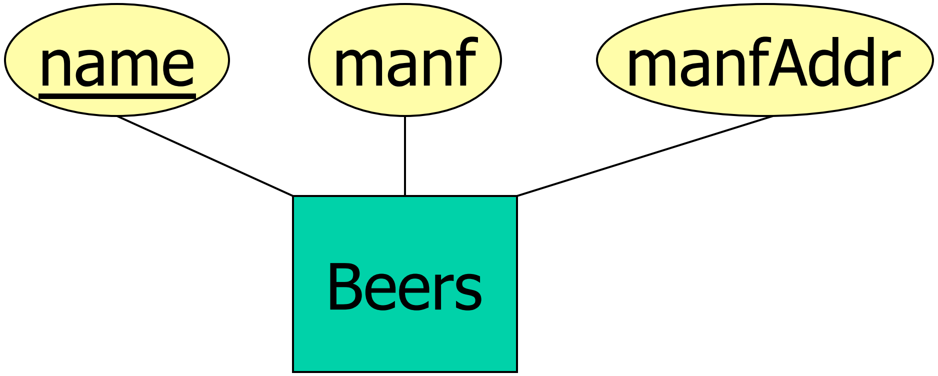
This design gives the address of each manufacturer exactly once

Example: Bad



This design states the manufacturer of a beer twice: as an attribute and as a related entity.

Example: Bad



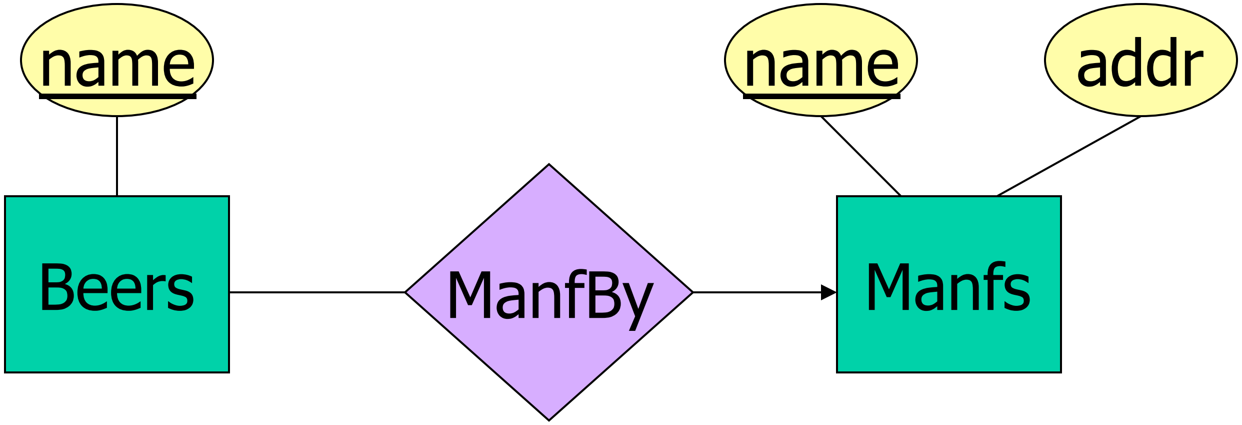
This design repeats the manufacturer’s address once for each beer and loses the address if there are temporarily no beers for a manufacturer

Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one nonkey attribute

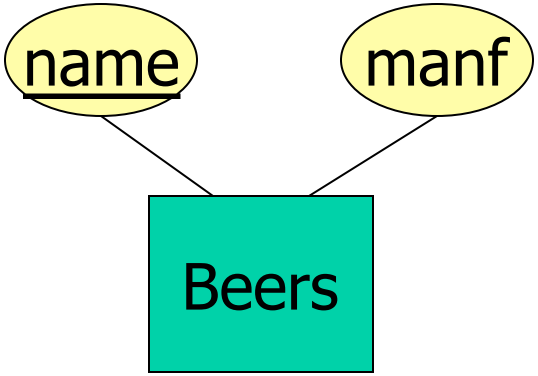
    or
  - It is the "many" in a many-one or many-many relationship

Example: Good



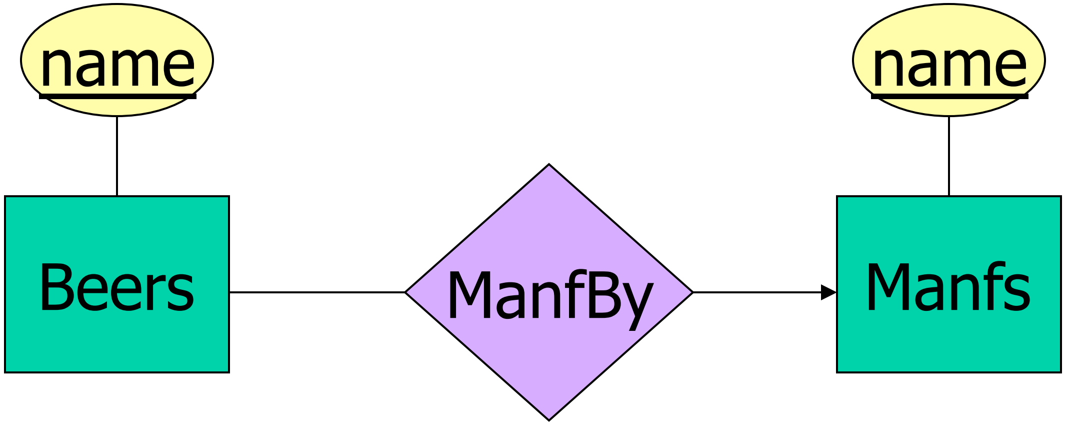
- Manfs deserves to be an entity set because of the nonkey attribute addr
- Beers deserves to be an entity set because it is the “many” of the many-one relationship ManfBy

Example: Good



There is no need to make the manufacturer an entity set, because we record nothing about manufacturers besides their name

Example: Bad



Since the manufacturer is nothing but a name, and is not at the "many" end of any relationship, it should not be an entity set

Don’t Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself
  - They make all entity sets weak, supported by all other entity sets to which they are linked
- In reality, we usually create unique ID’s for entity sets
  - Examples include CPR numbers, car’s license plates, etc.

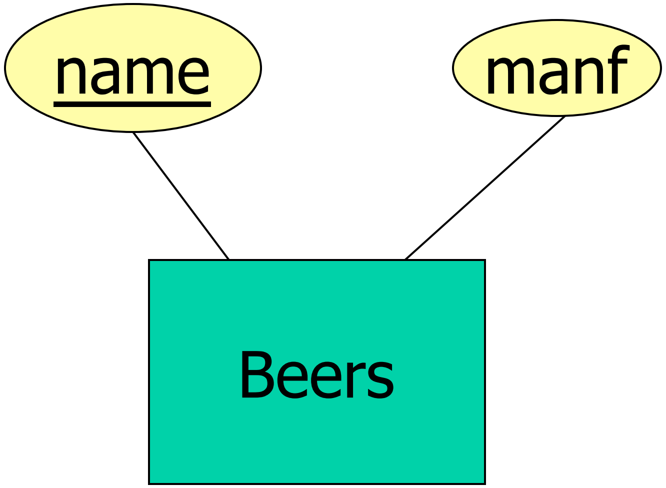
When Do We Need Weak Entity Sets?

- The usual reason is that there is no global authority capable of creating unique ID’s
- Example: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world

From E/R Diagrams to Relations

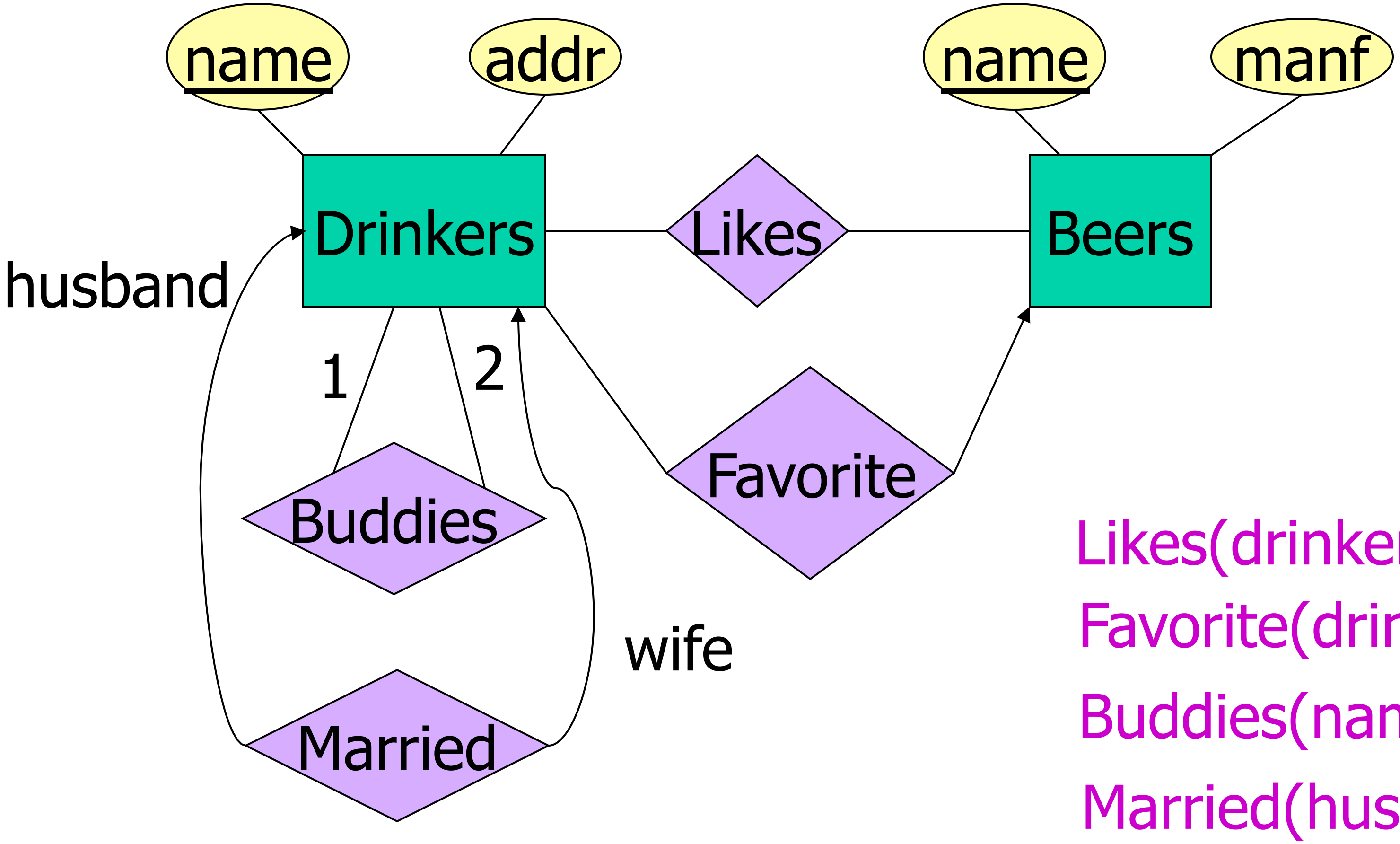
- Entity set → relation
  - Attributes → attributes
- Relationships → relations whose attributes are only:
  - The keys of the connected entity sets
  - Attributes of the relationship itself

Entity Set → Relation



Relation: Beers(name, manf)

Relationship → Relation



Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Combining Relations

- OK to combine into one relation:
  1. The relation for an entity-set *E*
  2. The relations for many-one relationships of which *E* is the "many"
- Example: Drinkers(name, addr) and Favorite(drinker, beer) combine to make Drinker1(name, addr, favBeer)

Risk with Many-Many Relationships

- Combining Drinkers with Likes would be a mistake. It leads to redundancy, as:

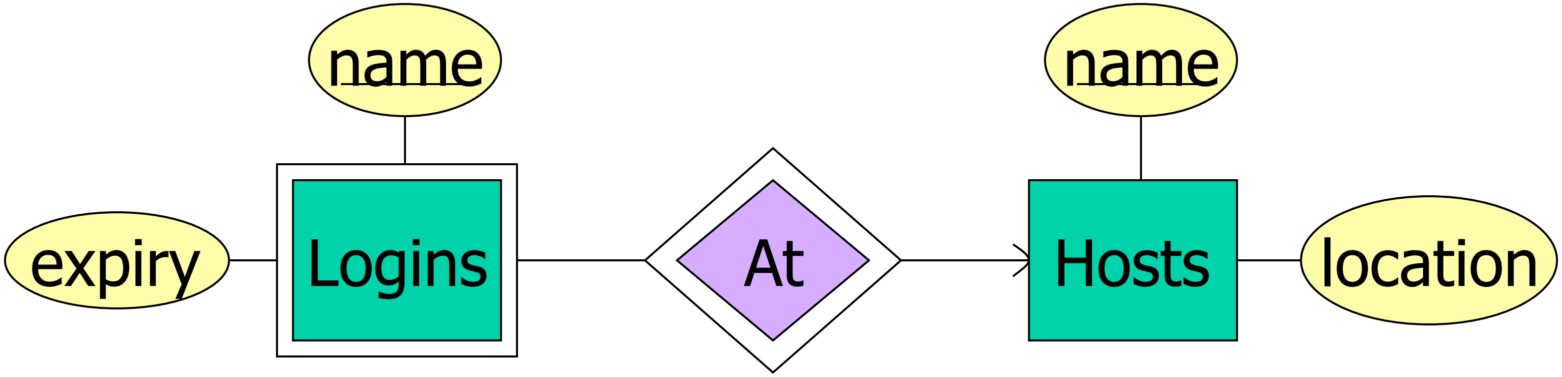
name	addr	beer
Peter	Campusvej	Od.Cl.
Peter	Campusvej	Erd.W.

Redundancy

Handling Weak Entity Sets

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes
- A supporting relationship is redundant and yields no relation (unless *it* has attributes)

Example: Weak Entity Set → Relation



Hosts(hostName, location)

Logins(loginName, hostName, expiry)

~~At(loginName, hostName, hostName2)~~

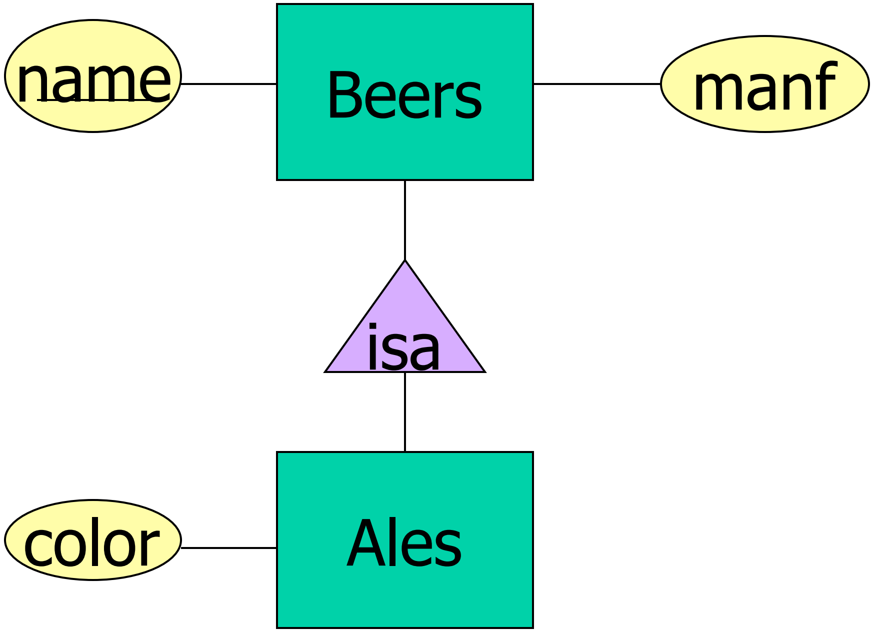
At becomes part of Logins

Must be the same

Subclasses: Three Approaches

1. *Object-oriented* : One relation per subset of subclasses, with all relevant attributes
2. *Use nulls* : One relation; entities have NULL in attributes that don't belong to them
3. *E/R style* : One relation for each subclass:
   - Key attribute(s)
   - Attributes of that subclass

Example: Subclass → Relations



Object-Oriented

name	manf
Odense Classic	Albani

Beers

name	manf	color
HC Andersen	Albani	red

Ales

Good for queries like “find the color of ales made by Albani”

E/R Style

name	manf
Odense Classic	Albani
HC Andersen	Albani

Beers

name	color
HC Andersen	red

Ales

Good for queries like "find all beers (including ales) made by Albani"

Using Nulls

name	manf	color
Odense Classic	Albani	NULL
HC Andersen	Albani	red

Beers

Saves space unless there are *lots* of attributes that are usually NULL

Summary 6

More things you should know:

- Entities, Attributes, Entity Sets,
- Relationships, Multiplicity, Keys
- Roles, Subclasses, Weak Entity Sets
- Design guidelines
- E/R diagrams → relational model

The Project

Purpose of the Project

- To try in practice the process of designing and creating a relational database application
- This process includes:
  - development of an E/R model
  - transfer to the relational model
  - normalization of relations
  - implementation in a DBMS
  - programming of an application

Project as part of The Exam

- Part of the exam and grading!
- The project must be done *individually*
- No cooperation is allowed beyond what is explicitly stated in the description

Subject of the Project

- *To create an electronic inventory for a computer store*
- Keep information about complete computer systems and components
- System should be able to
  - calculate prices for components and computer systems
  - make lists of components to order from the distributor

Objects of the System

- component: name, kind, price
  - kind is one of CPU, RAM, graphics card, mainboard, case
  - CPU: socket, bus speed
  - RAM: type, bus speed
  - mainboard: CPU socket, RAM type, on-board graphics?, form factor
  - case: form factor

Objects of the System

- computer system: catchy name, list of components
  - requires a case, a mainboard, a CPU, RAM, optionally a graphics card
  - sockets, bus speed, RAM type, and form factor must match
  - if there is no on-board graphics, a graphics card must be included

Objects of the System

- current stock: list of components and their current amount
- minimum inventory: list of components, their allowed minimum amount, and their preferred amount after restocking

Intended Use of the System

- Print a daily price list for components and computer systems
- Give quotes for custom orders
- Print out a list of components for restocking on Saturday morning (computer store restocks his inventory every Saturday at his distributor)

Selling Price

- Selling price for a component is the price + 30%
- Selling price for a computer system is sum of the selling prices of the components rounded up to next ’99‘
- Rebate System:
  - total price is reduced by 2% for each additional computer system ordered
  - maximal 20% rebate

Example: Selling Price

- computer system for which the components are worth DKK 1984
- the selling price of the components is 1984*1.3 = 2579.2
- It would be sold for DKK 2599
- Order of 3 systems: DKK 7485, i.e., DKK 2495 per system
- Order of 11, 23, or 42 systems: DKK 2079 per system

Functionality of the System

- List of all components in the system and their current amount
- List of all computer systems in the system and how many of each could be build from the current stock
- Price list including all components and their selling prices grouped by kind all computers systems that could be build from the current stock including their components and selling price

Functionality of the System

- Price offer given the computer system and the quantity
- Sell a component or a computer system by updating the current stock
- Restocking list including names and amounts of all components needed for restocking to the preferred level

Limitations for the Project

- No facilities for updating are required except for the Selling mentioned explicitly
- Only a simple command-line based interface for user interaction is required
  - Choices by the user can be input by showing a numbered list of alternatives or by prompting for component names, etc.
- You are welcome to include update facilities or make a better user interface but *this will not influence the final grade!*

Tasks

1. Develop an appropriate E/R model
2. Transfer to a relational model
3. Ensure that all relations are in 3NF (decompose and refine the E/R model)

---

4. Implement in PostgreSQL DBMS (ensuring the constraints hold)
5. Program in Java or Python an application for the user interaction providing all functionality from above

Test Data

- Can be made up as you need it
- At least in the order of 8 computer systems and 30 components
- Sharing data with other participants in the course is explicitly allowed and *encouraged*

Formalities

- Printed report of approx. 10 pages
  - design choices and reasoning
  - structure of the final solution
  - Must include:
    - A diagram of your E/R model
    - Schemas of your relations
    - Arguments showing that these are in 3NF
    - Central parts of your SQL code + explanation
    - A (very) short user manual for the application
    - Documentation of testing

Milestones

- There are two stages:
  1. Tasks 1-3, deadline March 11
     Preliminary report describing design choices, E/R model, resulting relational model
     (will be commented on and handed back)
  2. Tasks 4-5, deadline March 25
     Final report as correction and extension of the preliminary report
- Grade for the project will be based both on the preliminary and on the final report

Implementation

- Java with fx JDBC as DB interface
- Python with fx psycopg2 as DB interface
- SQL and Java/Python code handed in electronically with report in Blackboard
- Database for testing must be available on the PostgreSQL server
- Testing during grading will use your program and the data on that server