

ER Modelling

With slides by Ph.Bonnet, J.Ulmann

Outline

- Conceptual Model
 - ER notions
 - UML notation
- Good Design
 - Anomalies
 - Functional Dependencies
 - Normal Forms

Framework for Conceptual Model

- 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, and document its structure.

ER: Entity – Relationships

- Conceptual Model Defined by Peter Chen (1976)
- Concepts + Visual representation
- ER Not standardized
 - Every company has its own visual representation
 - Core concepts are universally accepted
- UML used as standard notation for ER model
 - Slight differences
 - “Applying UML and Patterns” - chapter 37 focuses on OR mapping (lecture 6) and data model (ER)

Entity

- *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.

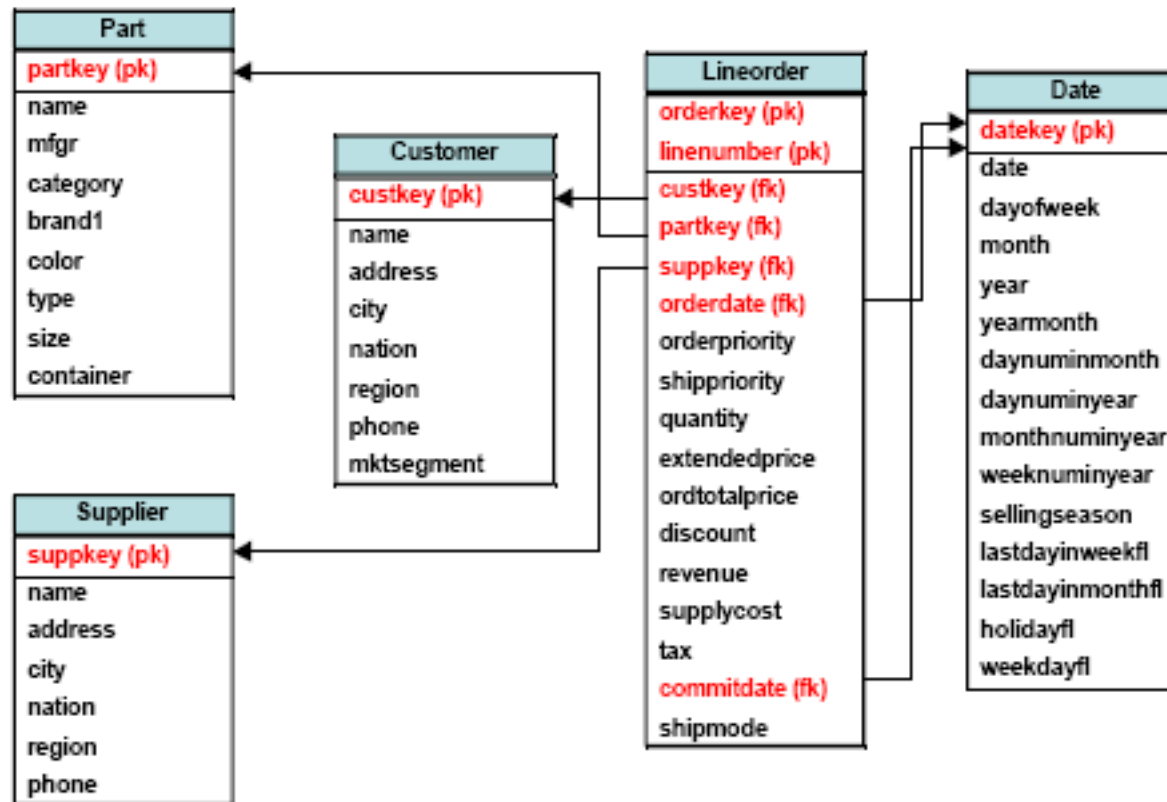
Relationships

- A **relationship** connects two or more entity sets.
- *Relationship set* = collection of similar relationships.
- *Attribute* = property of (the relationships of) a relationship set.

Unified Modelling Language

- UML is designed to model software, but has been adapted as a database modeling language.

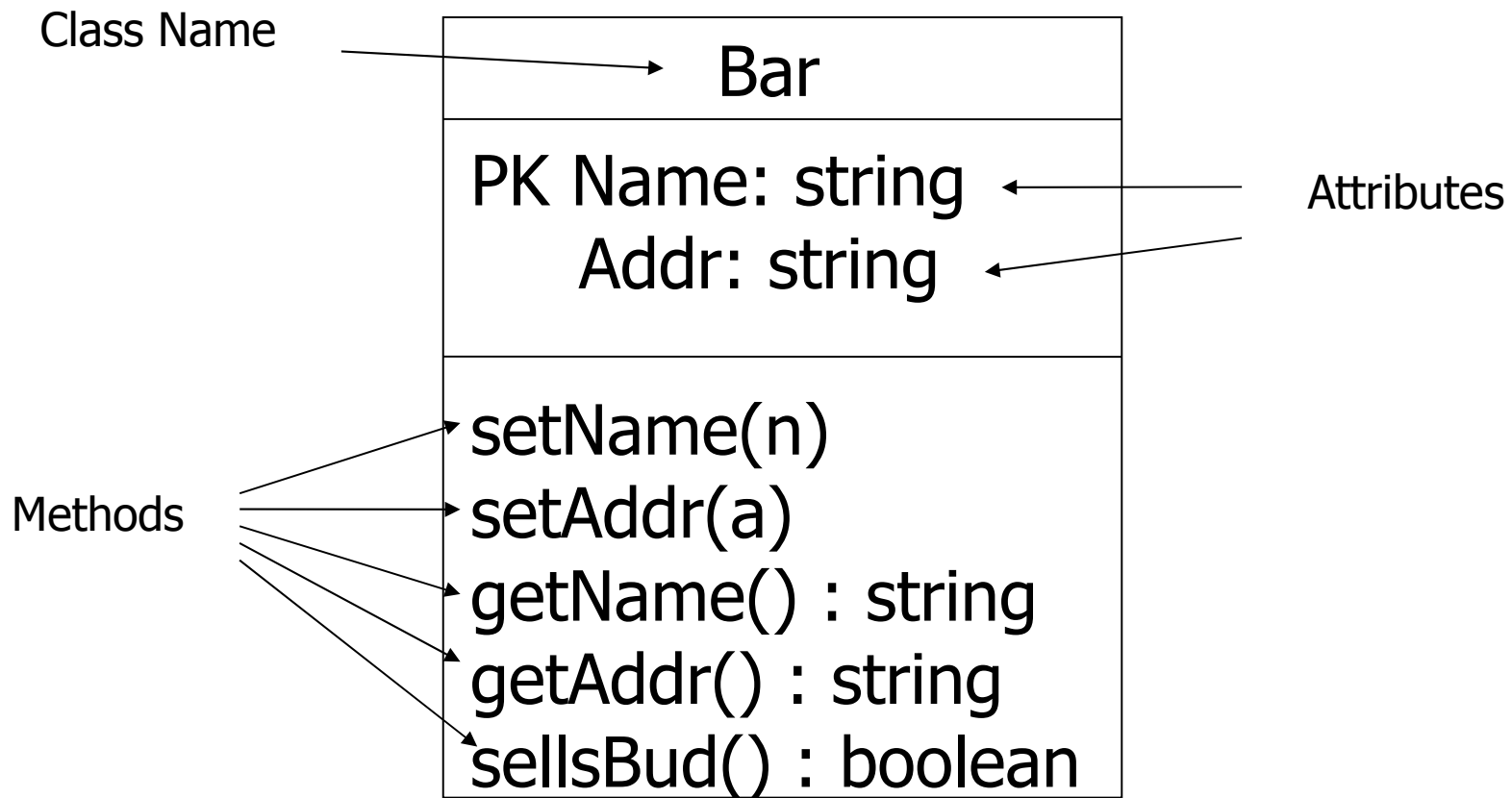
TPC-H SCHEMA



Classes

- Sets of objects, with attributes (*state*) and methods (*behavior*).
- Attributes have types.
- PK indicates an attribute in the primary key (optional) of the object.
- Methods have declarations: arguments (if any) and return type.

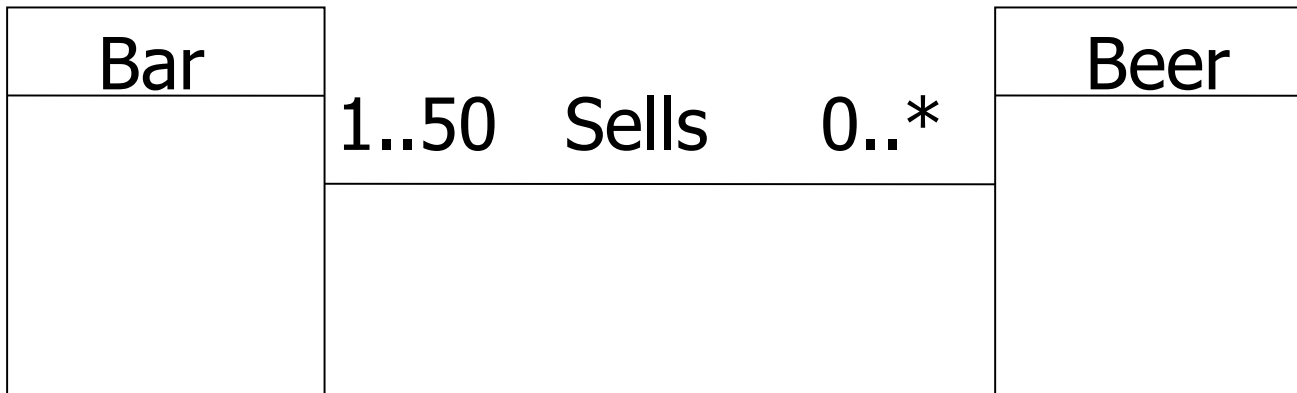
Example: Bar Class



Associations

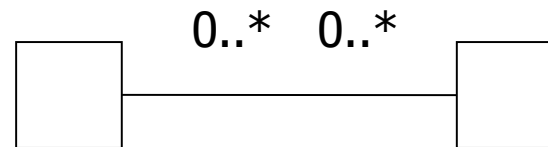
- **Binary** relationships between classes.
 - Beware the Kifer book introduces a diamond notation for associations between more than two classes. Ignore this and rely exclusively on binary relationships!
- Represented by named lines (role)
- Multiplicity at each end.
 - $m ..n$ means between m and n of these associate with one on the other end.
 - $*$ = “infinity”; e.g. $1..*$ means “at least one.”

Example: Association

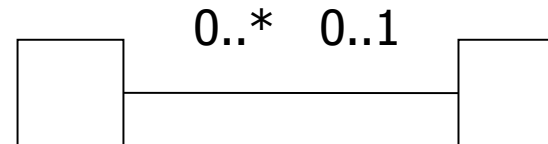


Interesting Multiplicities

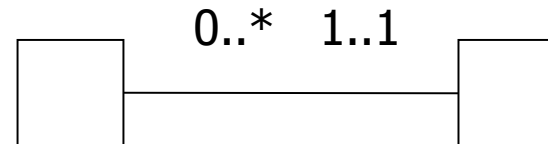
N – N relationship



At most 1 relationship



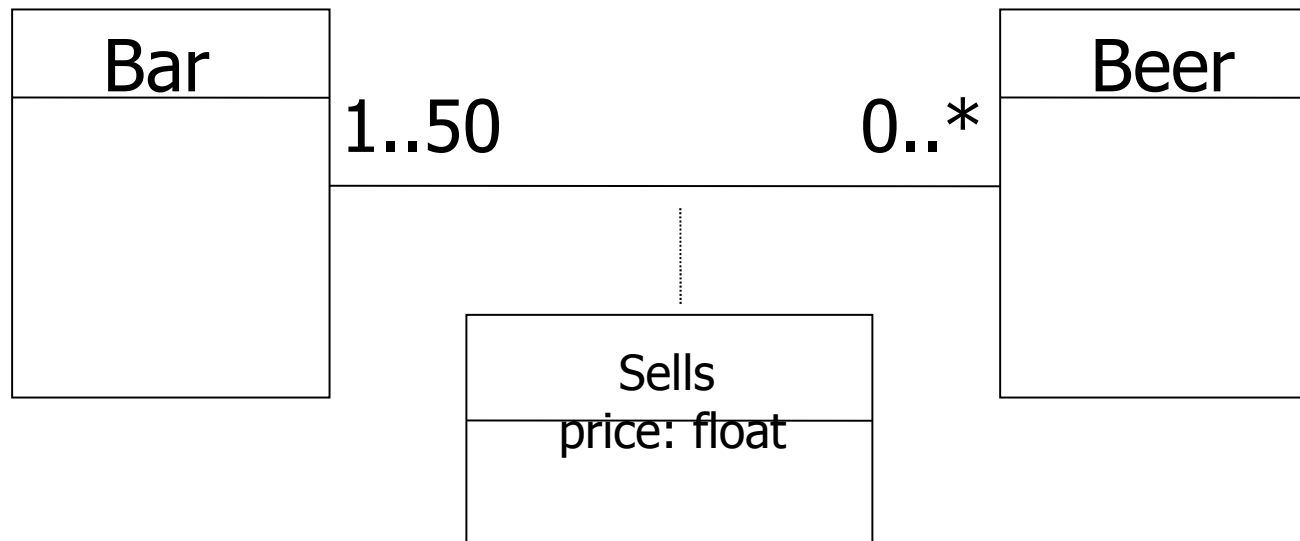
Exactly one relationship



Association Classes

- Attributes on associations are permitted.
 - Called an *association class*.

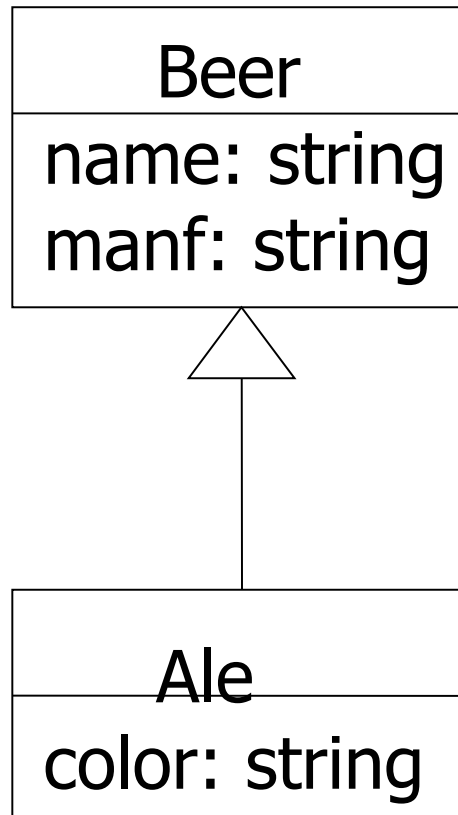
Example: Association Class



Subclasses

- Inheritance abstraction: subclass points to superclass with a line ending in a triangle.
- The subclasses of a class can be:
 - *Complete* (every object is in at least one subclass) or *partial*.
 - *Disjoint* (object in at most one subclass) or *overlapping*.

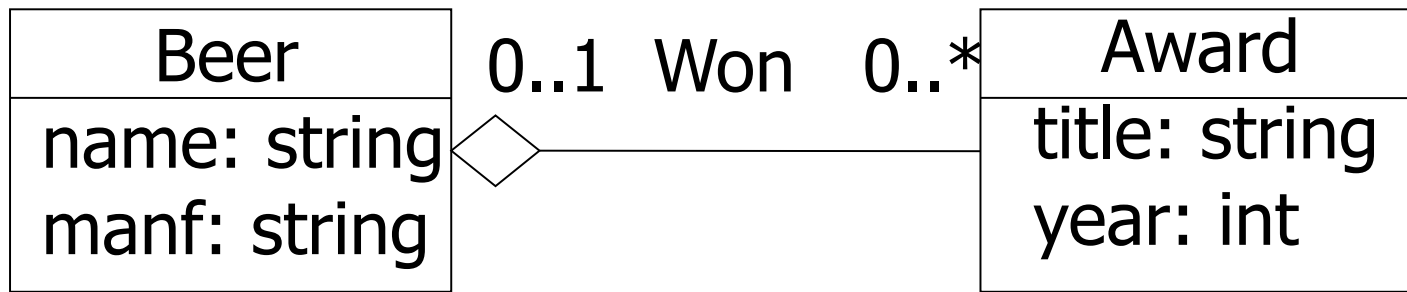
Example: Subclasses



Aggregations

- Relationships with implication that the objects on one side are “owned by” or are part of objects on the other side.
- Represented by a diamond at the end of the connecting line, at the “owner” side.
- Implication that in a relational schema, owned objects are part of owner tuples.

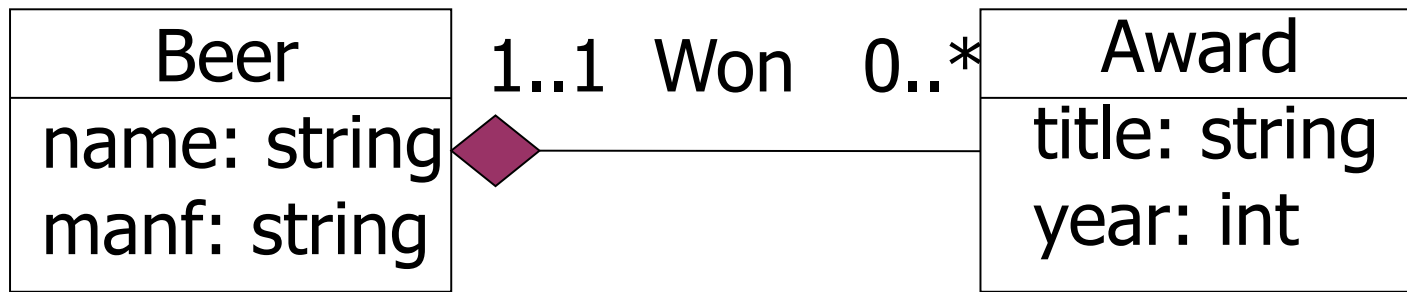
Example: Aggregation



Compositions

- Like aggregations, but with the implication that every object is definitely owned by one object on the other side.
- Represented by solid diamond at owner.
- Often used for subobjects or structured attributes.

Example: Composition



Conversion to Relations

- Each Class (entity) becomes a relation schema
 - Class name: relation name
 - Attributes: list of attributes with types
 - Key constraints

Conversion to Relations

- Problem with set-value attributes
 - *Straightforward Solution*: Use several rows to represent a single entity
 - (111111, John, 123 Main St, stamps)
 - (111111, John, 123 Main St, coins)
 - Problems with this solution:
 - Redundancy
 - Key of entity type (Id) not key of relation
 - The key of the relation is (Id, Hobby)
 - Hence, the resulting relation must be further transformed (Normalization)

Conversion to Relations

- We can use any of the following three strategies to convert a class and its subclasses to relations.
 1. E/R-style: each subclass' relation stores only its own attributes, plus key.
 2. OO-style: relations store attributes of subclass and all superclasses.
 3. Nulls: One relation, with NULL's as needed.

E/R Style

name	manf
Bud Summerbrew	Anheuser-Busch Pete's

Beers

name	color
Summerbrew	dark

Ales

Good for queries like
"find all beers (including
ales) made by Pete's."

Object-Oriented

name	manf
Bud	Anheuser-Busch

Beers

name	manf	color
Summerbrew	Pete's	dark

Ales

Good for queries like "find the color of ales made by Pete's."

Using Nulls

name	manf	color
Bud Summerbrew	Anheuser-Busch Pete's	NULL dark

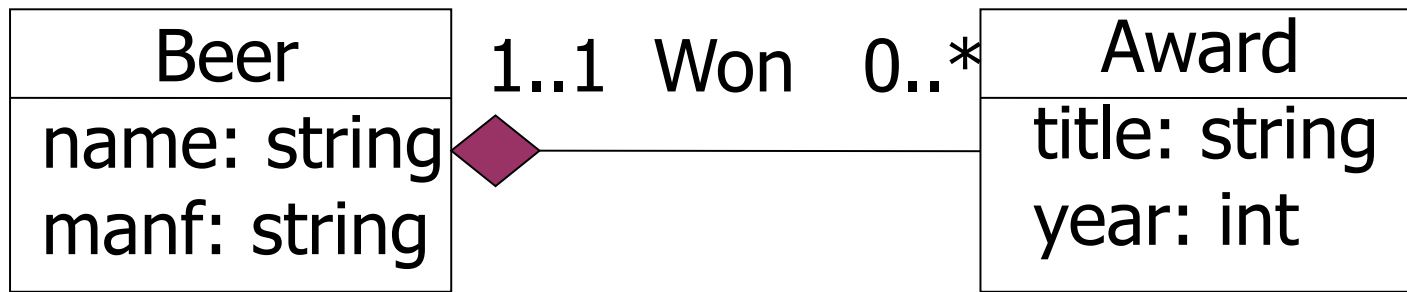
Beers

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

Conversion to Relations

- Each association/association class (relationship) becomes a relation
 - Relation name is role name
 - Attributes of association class
 - For each role, the primary key of the entity type associated with that role
- Specific cases for composition/aggregation

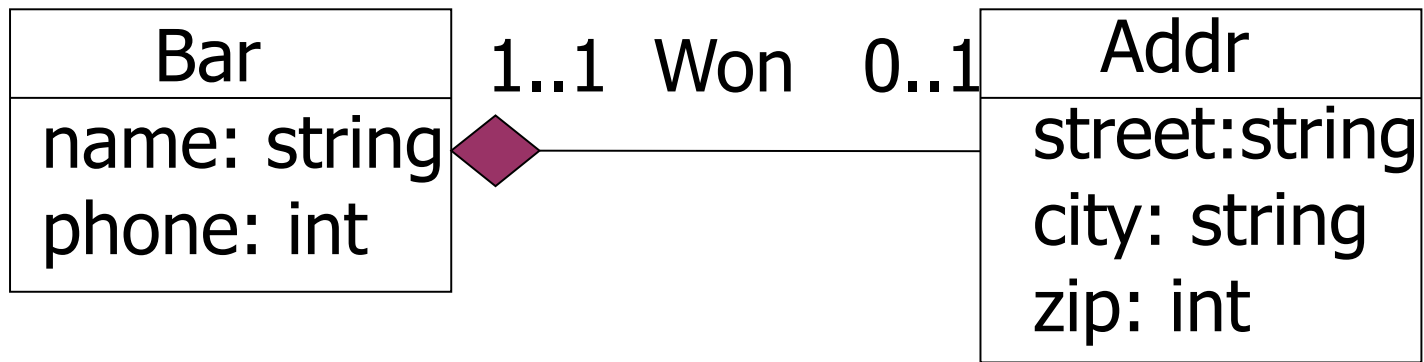
Example: Composition



Conversion to Relations: Composition

- We must store the awards as a separate table – same case as multivalued (e.g., set/list) attributes.

Example: Composition



Conversion to Relations: Composition

- Since a bar has at most one address, it is quite feasible to add the street, city, and zip attributes of Addr to the Bars relation.

Outline

- Conceptual Model
 - ER notions
 - UML notation
- Good Design
 - Anomalies
 - Functional Dependencies
 - Normal Forms

Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy.

Update anomaly : one occurrence of a fact is changed, but not all occurrences.

Deletion anomaly : valid fact is lost when a tuple is deleted.

Bad Design Example

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

name	address	beersLiked	manf	favBeer
Janeway	Voyager	Bud	AB	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	AB	Bud

- **Update anomaly:** if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- **Deletion anomaly:** If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Functional Dependencies

- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X , then they must also agree on all attributes of Y .
 - Say “ $X \rightarrow Y$ holds in R .”
 - **Convention:** ..., X, Y, Z represent sets of attributes; A, B, C, \dots represent single attributes.
 - **Convention:** no set formers in sets of attributes, just ABC , rather than $\{A, B, C\}$.

Splitting Right Sides of FD's

- $X \rightarrow A_1 A_2 \dots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_n$ hold for R .
- **Example:** $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.
- There is no splitting rule for left sides.
- We'll generally express FD's with singleton right sides.

Example: FD's

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

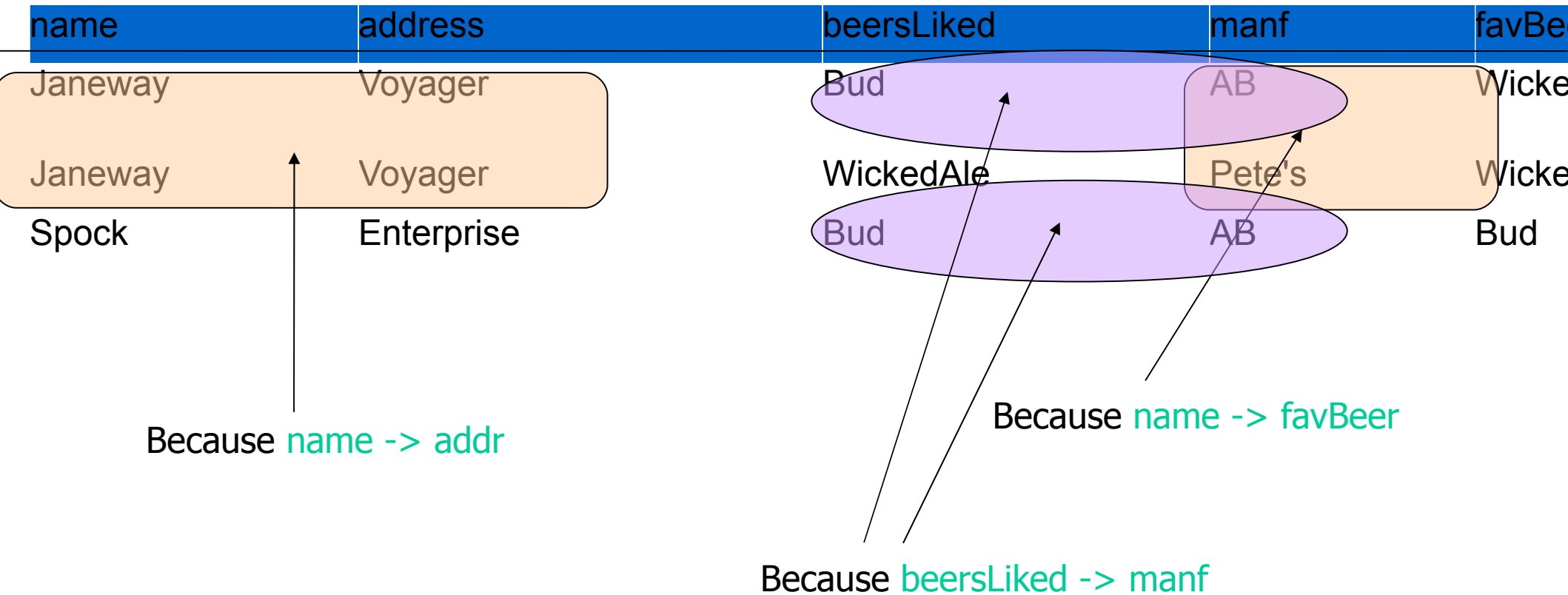
- Reasonable FD's to assert:

name -> addr, favBeer

- Note this FD is the same as name -> addr and name -> favBeer.

beersLiked -> manf

Example: Possible Data



Keys of Relations

- ◆ K is a *superkey* for relation R iff K functionally determines all of R .
- ◆ K is a *key* for R if K is a superkey, but no proper subset of K is a superkey.

Example: Superkey

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

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes.
 - name -> addr favBeer
 - beersLiked -> manf

Example: Key

- {name, beersLiked} is a **key** because neither {name} nor {beersLiked} is a superkey.
 - name \rightarrow manf does not hold
 - beersLiked \rightarrow addr does not hold
- There are no other keys, but lots of superkeys.
 - Any superset of {name, beersLiked}.

Where Do Keys Come From?

1. Just assert a key K .
 - ▶ The only FD's are $K \rightarrow A$ for all attributes A .
1. Assert FD's and deduce the keys by systematic exploration.

More FD's From "Physics"

- ◆ **Example:** "no two courses can meet in the same room at the same time" tells us:
hour, room -> course.

Inferring FD's

- ◆ We are given FD's $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation that satisfies the given FD's.
 - ◆ Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don't say so.
- ◆ Important for design of good relation schemas.
 - ◆ Closure of a set of attributes X , noted X^+ : all FDs where X on the left hand side.

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.
 - ▶ *nontrivial* means Y is not contained in X .
 - ▶ Remember, a *superkey* is any superset of a key

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.

Example: BCNF Decomposition

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

$F = \text{name} \rightarrow \text{addr}, \text{name} \rightarrow \text{favBeer}, \text{beersLiked} \rightarrow \text{manf}$

- ◆ Pick BCNF violation $\text{name} \rightarrow \text{addr}$.
- ◆ Close the left side: $\{\text{name}\}^+ = \{\text{name}, \text{addr}, \text{favBeer}\}$.
- ◆ Decomposed relations:
 - ▶ Drinkers1(name, addr, favBeer)
 - ▶ Drinkers2(name, beersLiked, manf)

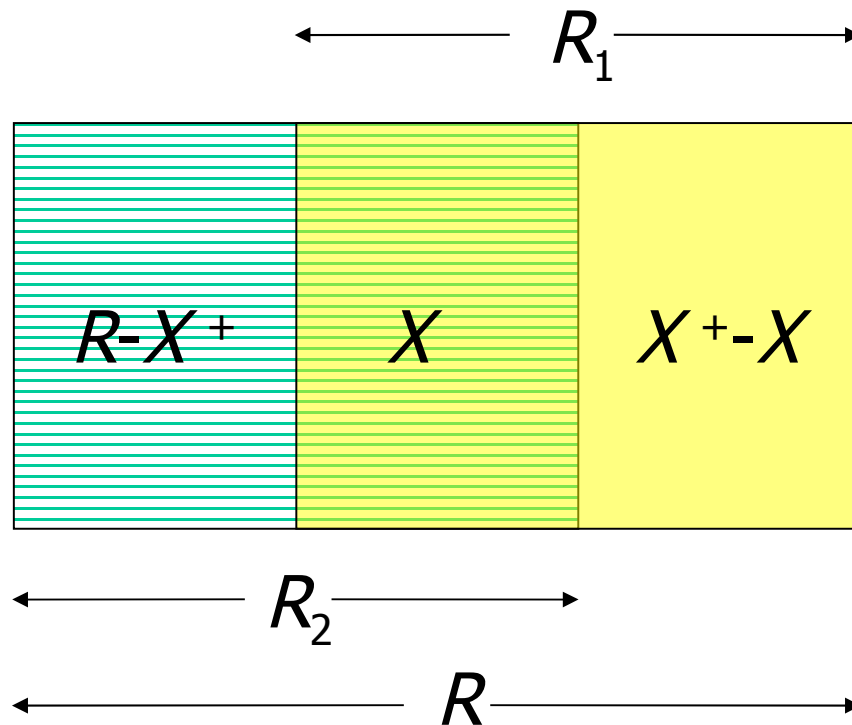
Example -- Continued

- ◆ 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 -- Continued

- ◆ For $Drinkers2(\underline{name}, \underline{beersLiked}, manf)$, the only FD is $beersLiked \rightarrow manf$, and the only key is $\{name, beersLiked\}$.
 - ▶ Violation of BCNF.
- ◆ $beersLiked^+ = \{beersLiked, manf\}$, so we decompose $Drinkers2$ into:
 - ▶ $Drinkers3(\underline{beersLiked}, manf)$
 - ▶ $Drinkers4(\underline{name}, \underline{beersLiked})$

Decomposition Picture



Example -- Concluded

- ◆ The resulting decomposition of *Drinkers* :
 - ◆ Drinkers1(name, addr, favBeer)
 - ◆ Drinkers3(beersLiked, manf)
 - ◆ 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.

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 = zip 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 $A = \text{street}$, $B = \text{city}$, and $C = \text{zip}$ on the next slide.

An Unenforceable FD

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

Join tuples with equal zip codes.

street	city	zip
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations, FD **street, city -> zip** 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-zip is an example.

Take Away Points

- UML Design

- Entities and relationships mapped onto classes and associations
- Multiplicity to indicate referential integrity constraints
- Subclasses, composition and aggregation to capture model characteristics

- Conversion to relations

- Classes and associations mapped to relations
 - Multivalued attributes introduce redundancy (see Good Design)
 - Special cases for subclasses, composition

Take-Away Points

Good Design

Avoid update / deletion anomalies

Functional Dependencies

Define key constraints

Drive relation decomposition (foreign keys)

Relation decomposition into 3NF or BCNF

Normalize to avoid update/deletion anomalies

Might denormalize for performance