Introduction to Databases

Motivation
Historical
Classical Models
E/R Model

Robert Keller

21 January 2004

Definition: Database

- A collection of data items
  - structured so as to provide efficient access to the items based on partial knowledge of those items, and
  - stored persistently and independently of the application(s) accessing those items.

Typical Characteristics of Databases

- Large volume of data
- Persistent storage
- Non-volatile media (vs. main memory)
- Logging and recovery
- Access control
- Concurrency control
- Transaction-oriented
- Languages for querying and updating

Definition: Enterprise

- Databases are not just things for storing data.
- They are also a **model** of a set of **real-world** activities, the running of which they are intended to facilitate.
- These activities are collectively called the "**Enterprise**". (The term is used even for non-business applications.)

Definition: Database System

- A computer application or related set of applications that makes essential use of a database.

Examples of Database Systems

- Business data
  - Payroll, orders, reservations, catalogs
- Genealogies
- Legal cases, patents
- Library catalogs
- Meeting schedules
- Museum collections
- Music collections
- Scientific experiment results
- Software code versions
- Trouble reports and resolutions

Definition: Database Management System (DBMS)

- A software system that provides infrastructure for creating database systems.
- Infrastructure:
  - Query language(s)
  - API
  - Operating system/file system interfaces
  - Mechanisms for transactions, recovery, logging, etc.

Examples of Database Management Systems

- Oracle
- DB2 (IBM)
- MySQL
- PostgreSQL
- ObjectStore
- Poet

Definition: Database Administrator

- Someone endowed with special powers for a database system:
  - Create new user.
  - Set priveleges.
  - Modify the schema.

Definition: Schema

- The structural specification for a database.
- In some cases, the schema itself can be stored as a meta-database.
- Usually there is a special language for defining the schema, called a **Data Definition Language** (DDL), in contrast to the **Data Manipulation Language** (DML).

Related Areas

- Data Structures
- Logic
- Software Engineering
- Operating Systems (including distributed)
- Information Retrieval
- Data Mining, machine learning, inductive logic programming

Societal Issues

- Privacy
- Copyright, Intellectual Property
- Misinformation
- Legal liability
- Global Economy
- Standardization

Prehistory

- Punched cards, paper tape
- Magnetic tape

Paleozoic Era

- Data is in one or more files.
- Files are loaded entirely, or piecemeal, into main memory.
- Files are rewritten to commit changes.
- What problems do you see with this approach?

Mesozoic Era

- Sequential file, say ordered by key field
- Indexed sequential file, disks
- "Inverted" file
  (keys on other fields)

Cenozoic Era

- Data resides in directly-accessible non-volatile storage hierarchy.
- Units of data are moved to main memory on demand, then written back to storage unit by unit as the application progresses.

A Fundamental Dichotomy

- **Physical** access (where the data are on disk, how they are connected, how are they transported, etc.)
- **Logical** structure (what are the essential logical connections between data units)

Evolution of Database Models

- Early models (ca 1960's)
  - Hierarchical (IBM)
  - Navigational or "Network" (Codasyl DBTG)
- Relational (1969 to present)
- Entity/Relationship model (1976)
- Deductive databases (1980's)
- Object-Oriented DB's (1990's to present)
- World-wide-web
- Peer-to-peer databases
- XML databases

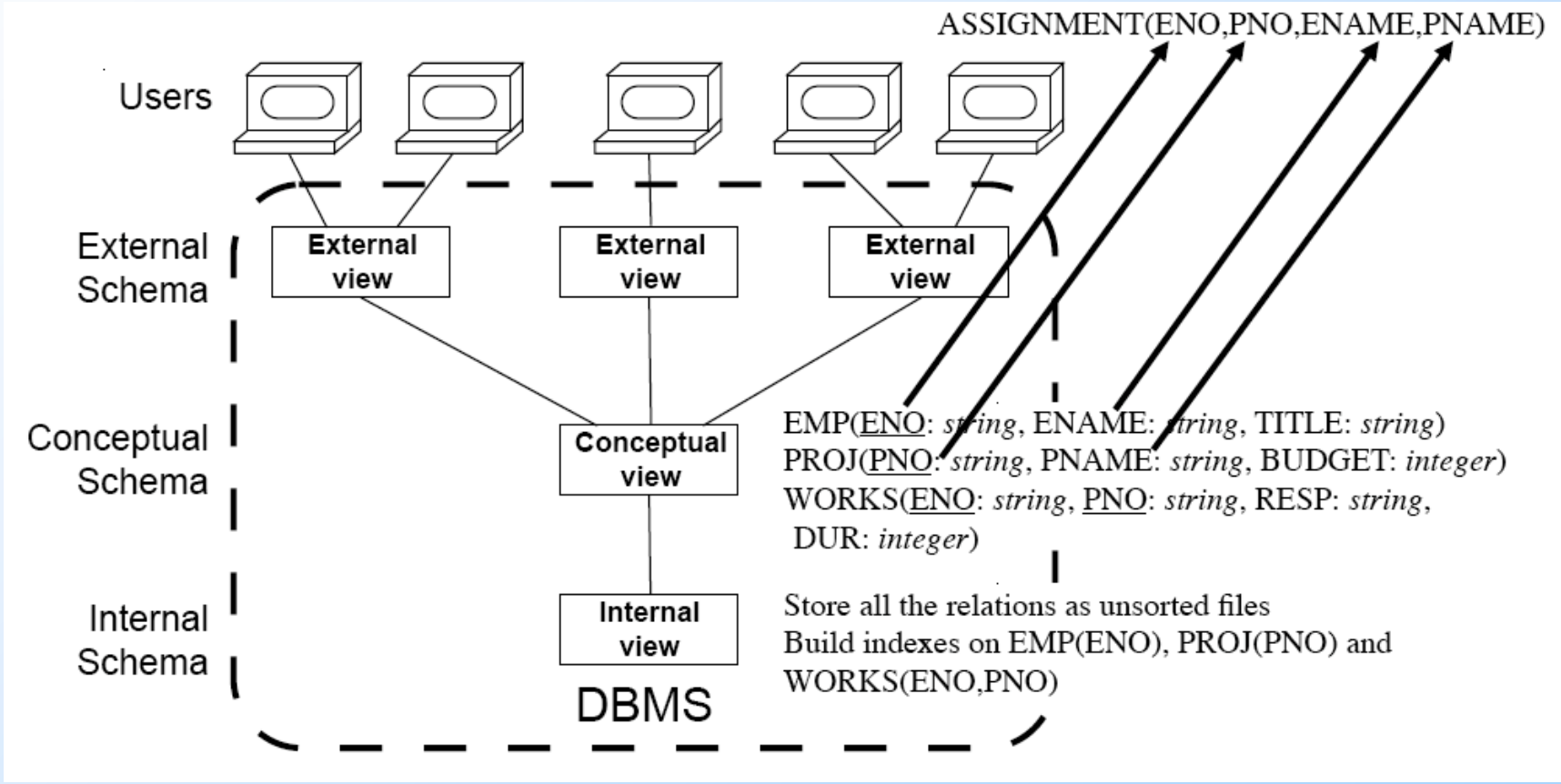
Main Tensions

- Provide human ease in structuring and querying, while simultaneously providing efficient access and integration with applications.
- Provide multiple, distributed, access paths, while simultaneously maintaining security and integrity of data.

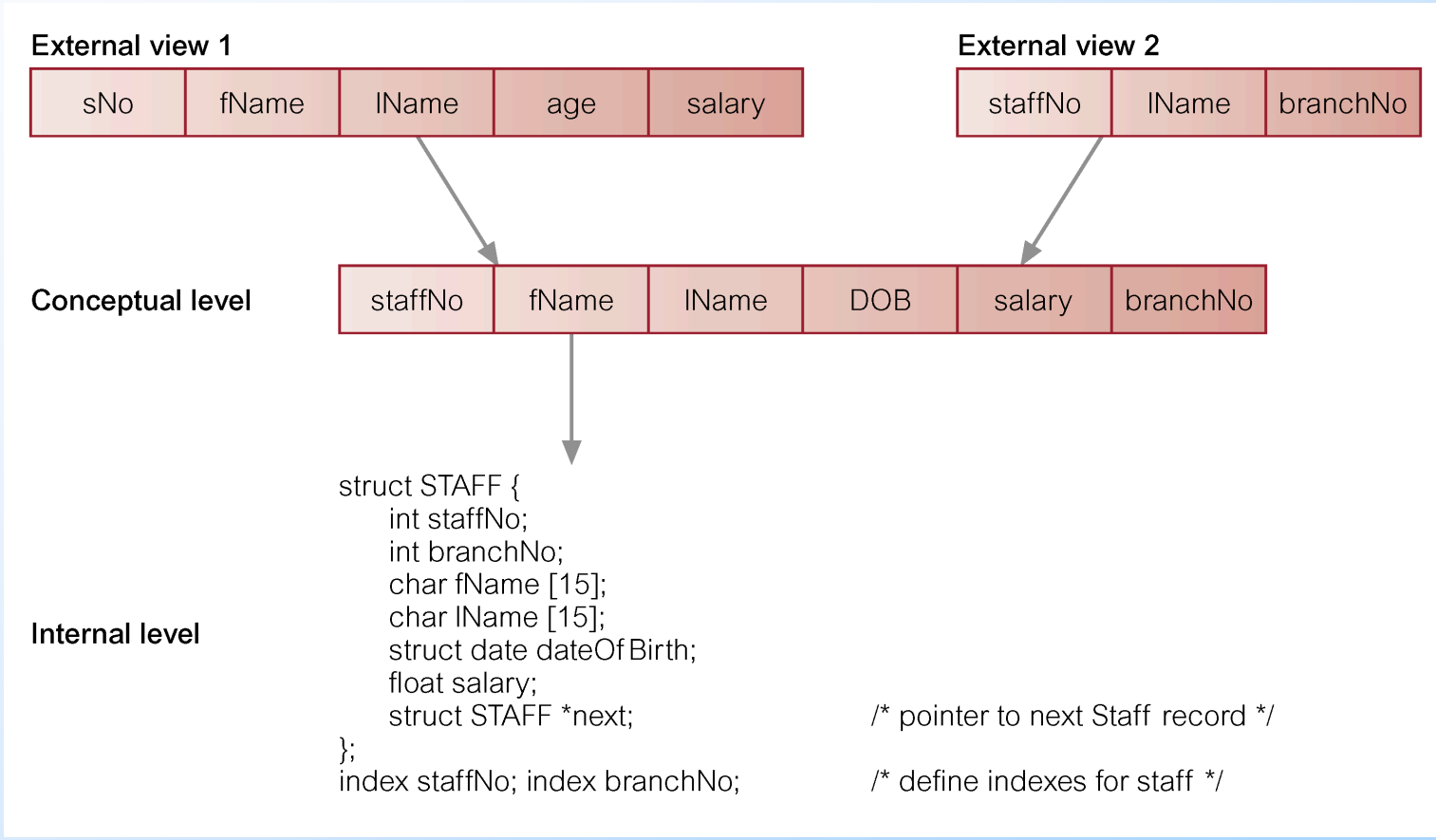
Early Attempt at Clarification: ANSI/SPARC Model (1977)

- ANSI = "American National Standards Institute"
- SPARC = "Standards Planning And Requirments Committee" (of ANSI)
- Clear separation of logical "views" from physical structure:
  - Internal view (physical, **changeable**)
  - Conceptual view (logical, system-wide, **stable**)
  - External views (logical, **per-application**)
- Proposal, rather than an actual standard, but still used as a software pattern, e.g. IHFS (Integrated Hydrologic Forecast System)

ANSI/SPARC Architecture



Differences between Three Levels of ANSI-SPARC Architecture



Classical Data Models

- Hierarchical Model
- Navigational or "Network" Model
- Relational Model

The Hierarchical Data Model

- Data is organized as a **forest**, with records as nodes.
- A record can contain **repeating groups**, each consisting of multiple records of another type.
- For example, a **department** record could have a repeating group **projects, projects** could have a repeating group **employee**, etc.

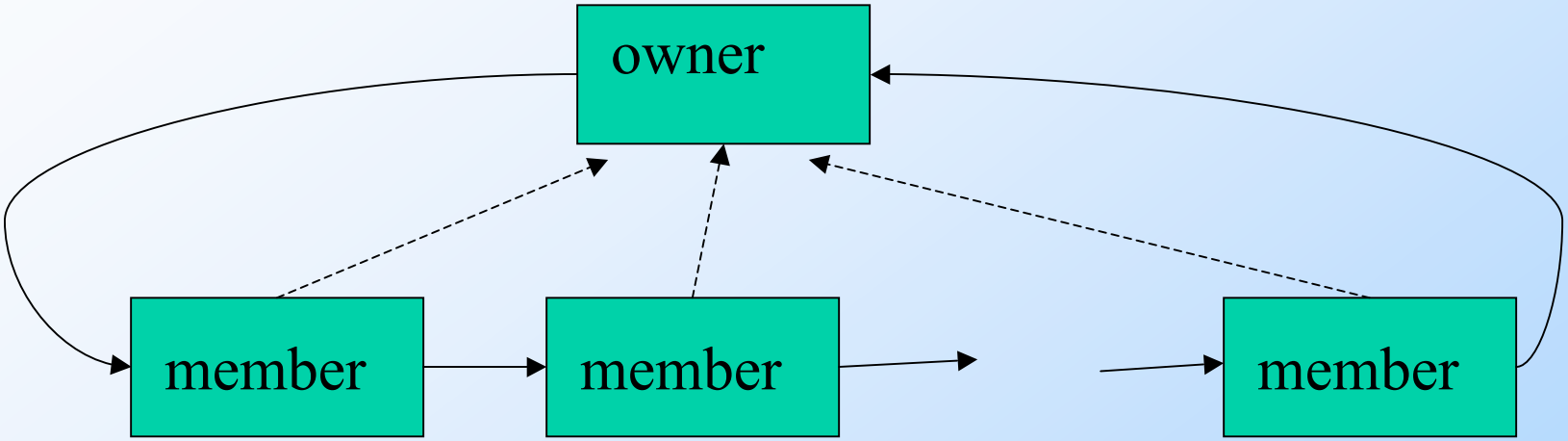
Instances of the Hierarchical Model

- IBM's IMS (Information Management System, 1968, still in use with System 390) was perhaps the original
  - Allowed virtual links, similar to unix file system, so not strictly a hierarchy
  - Accessibility integrity: No record could exist without its parent also existing.
  - Tutorial
- MUMPS (used in medical centers)
- Filemaker Pro (used in HMC administration)

The Navigational or "Network" Model

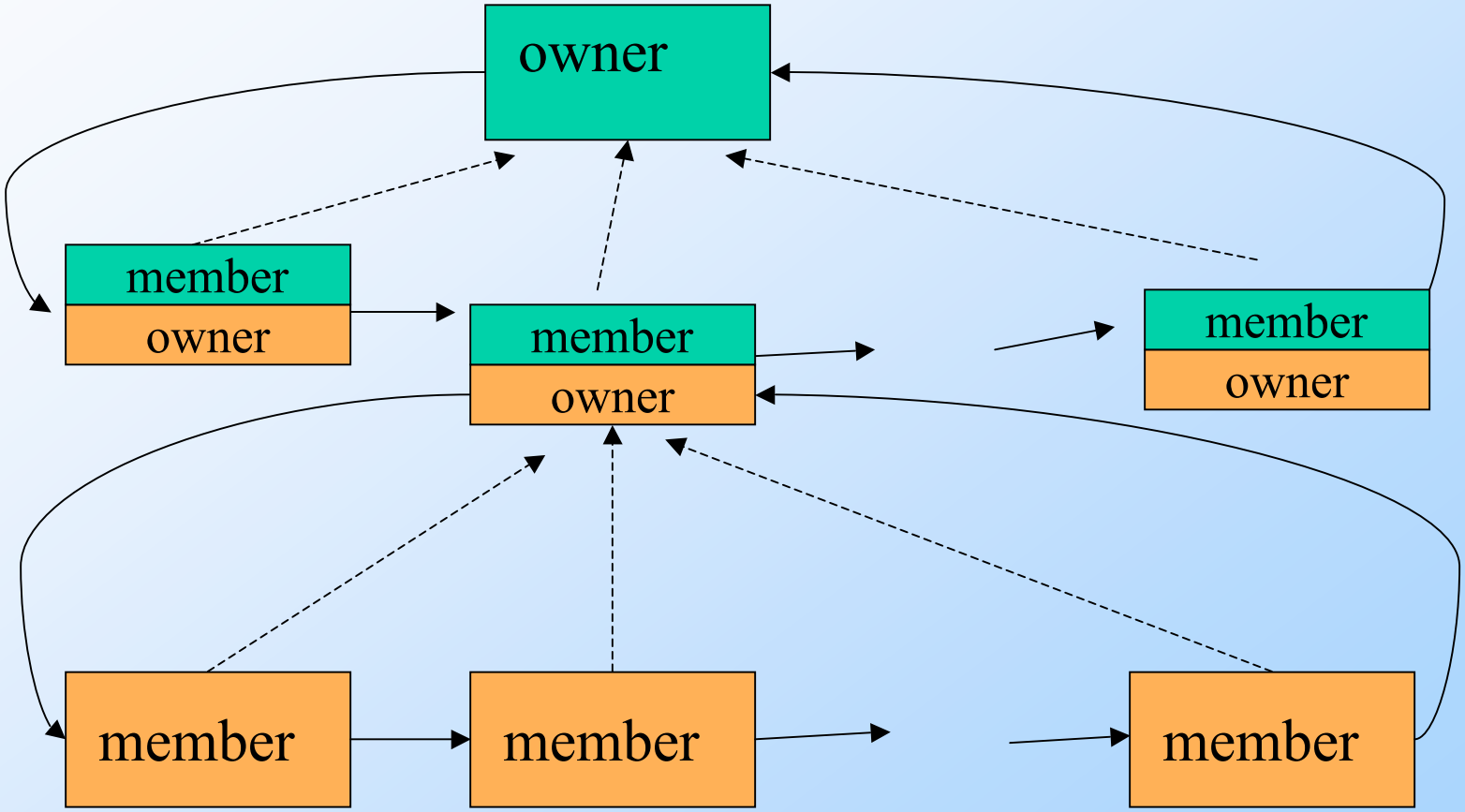
- A fore-runner of Object-Oriented Databases
- Data stored in records, which were strongly-typed.
- Records grouped together in "sets".
- Each "set" was like a circular list and had an owner record.
- Through commands, the programmer can "navigate" the database, by
  - Moving from one member of a set type to the next.
  - Moving from a member to the owner.
- Links provided a way to get from one type of set to another.
- Linked lists were the obvious physical model.

The Navigational or "Network" Model



The records themselves are stored on disk. There could be one current member of a given type set in memory.

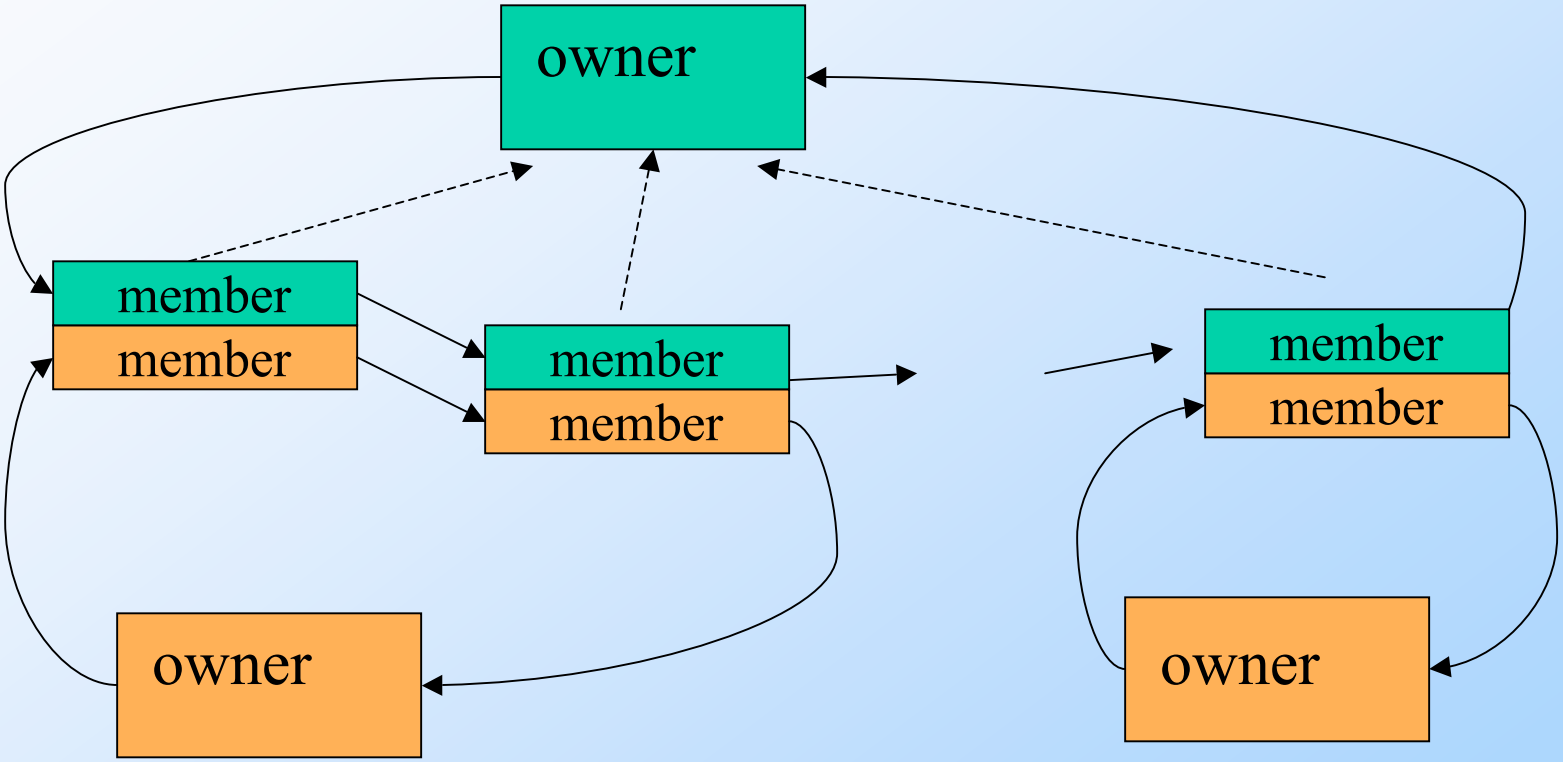
Each member of a set of one type could be the owner of a set of another type, and son on.



Intersecting tree-like hierarchies

- A record can be a member of at most one set of a given type, but it can be a member of other sets of different types.
- Consequently, the data organization, while somewhat hierarchical, is not strictly a tree.

Intersecting tree-like hierarchy



Navigation

- Through a process of navigation, the programmer would fetch current records of one or more types, collecting items for the response to a query.
- Explicit pointers were avoided by having fetch of the next record be relative to current records.
- The model was interfaced to specific programming languages (such as COBOL, Fortran).
- The process is fascinating, but ultimately somewhat cumbersome.

The Navigational or "Network" Model

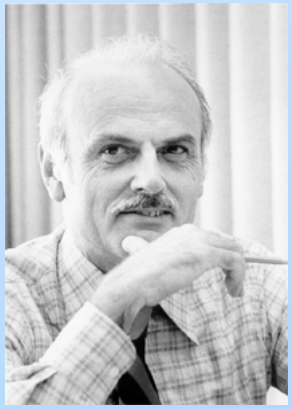
The Programmer as Navigator

by Charles W. Bachman

Charles Bachman initiated the DBTG.
He received the ACM Turing Award in 1973.

The Relational Model

- Relational is the dominant database model.
- Introduced by Edgar ("Ted") Codd of IBM in 1969. Codd received the Turing Award in 1981.



Edgar Codd, 1923-2003

A Few of Codd's 12 Rules

- *Rule 1: The Information Rule*
  All data should be presented to the user in table form.
- *Rule 2: Guaranteed Access Rule*
  All data should be accessible without ambiguity.
- *Rule 3: Systematic Treatment of Null Values*
  A field should be allowed to remain empty.
- *Rule 8: Physical Data Independence*
  The user should be isolated from the physical method of storing and retrieving information from the database.

The Relational Model

- Completely separates logical from physical.
- Data are stored in **relations** ("tables").
- Each table is a *set* of **tuples**.
- Queries based on predicate calculus or a functional formulation known as "relational algebra".
- Note that using logic for queries was in use within AI in 1968 (Cordell Green) or earlier.

Sample Relational DB

Computer relation

PropNo	Make	Model	AcqDate	Memory	HD
7	SGI	PowerFlop	4/1/1999	256M	10G
13	IBM	MegaSwap	12/31/2000	128M	100G
42	Dell	MediaBuster	6/12/2002	512M	30G

Location relation

PropNo	Bldg	Room	PrincUser
7	Olin	3006	Sam
13	Beckman	2001	Alice
42	Linde	8192	Leslie

More Relational Nomenclature

- The column headings are (names of) **Attributes**.

- A query that makes use of attributes common to multiple tables is called a **Join** query:

  "Find the Bldg and Room of all computers made by Dell."

Advantages of Relations

- Very general: functional data is a special case of relational.
- Fairly neutral about what kind of data are values of attributes.
- All types of attributes treated uniformly.

Perceived Difficulties with Relations

- All types of attributes are treated uniformly. So need to be prepared for queries that might **not** be likely or **meaningful**.
- Some queries can be very **expensive** to evaluate, depending on how the relations are set up initially.
- Unless the database is ***normalized***, which requires some care in design, there can be much replication of information and other bad things ("anomalies"). The database might not be a good model of the enterprise it is intended to serve.

The Entity/Relationship Model


Peter Chen

- This model was introduced in 1976 by Peter Chen, a professor at Louisiana State Univ.
- Its purpose was to make the connection with the enterprise clearer.
- It also was to be an implementation-neutral model that could be used for relational, navigational, hierarchical, and possibly other models.
- The UML of software engineering was evidently influenced by the E/R model, but E/R is not strictly a subset of UML.

Use of the E/R Model

- While it is easy to create a relational database, it is also easy to "get it wrong".
- The E/R model is typically used as the basis for a rational **database design** document.
- There is a graphical notation for it.

Ullman Slides

Many of the following slides are from Prof. Jeffrey Ullman, the first author of our text.

The original version of these slides is available on the web with audio voice-over.

I have freely made modifications to suit my own tastes in presentation.

Purpose of the E/R Model

- The E/R model allows us to sketch the design of a database informally.
- Designs are pictures called *entity-relationship diagrams*.
- Fairly mechanical ways to convert E/R diagrams to real implementations like relational databases exist.

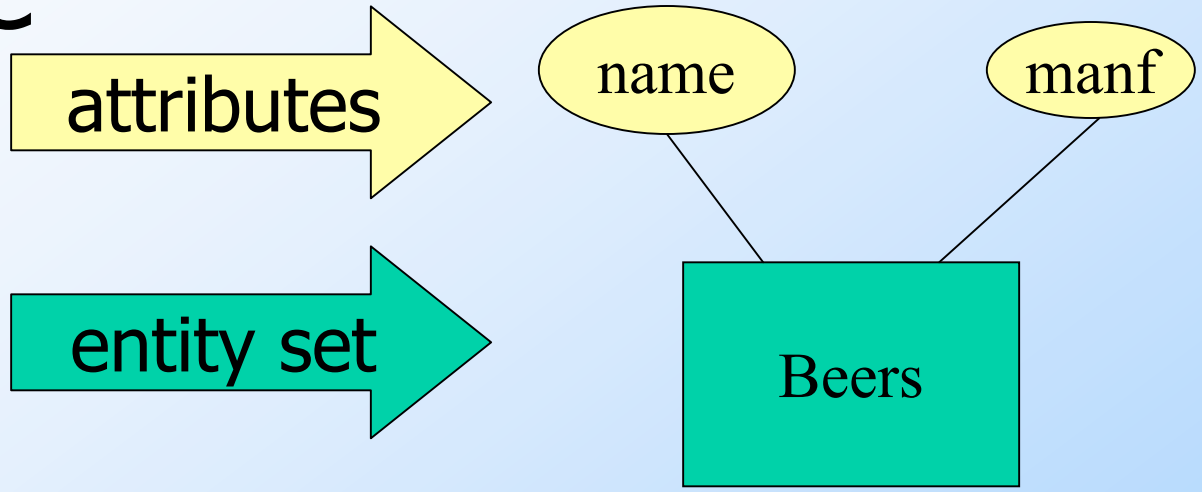
Entity Sets

- *Entity* = "thing" or object.
- *Entity set* = collection of similar entities.
  - Similar to a class in object-oriented languages.
- Attribute = property of an entity.
  - Generally, all entities in a set have the same types of attributes.
  - Attributes have simple values, e.g. integers, character strings, dates, images.

E/R Diagrams

- In an entity-relationship diagram, each **entity set** is represented by a **rectangle**.
- Each **attribute** of an entity set is represented by an **oval**, with a line to the rectangle representing its entity set.

Example



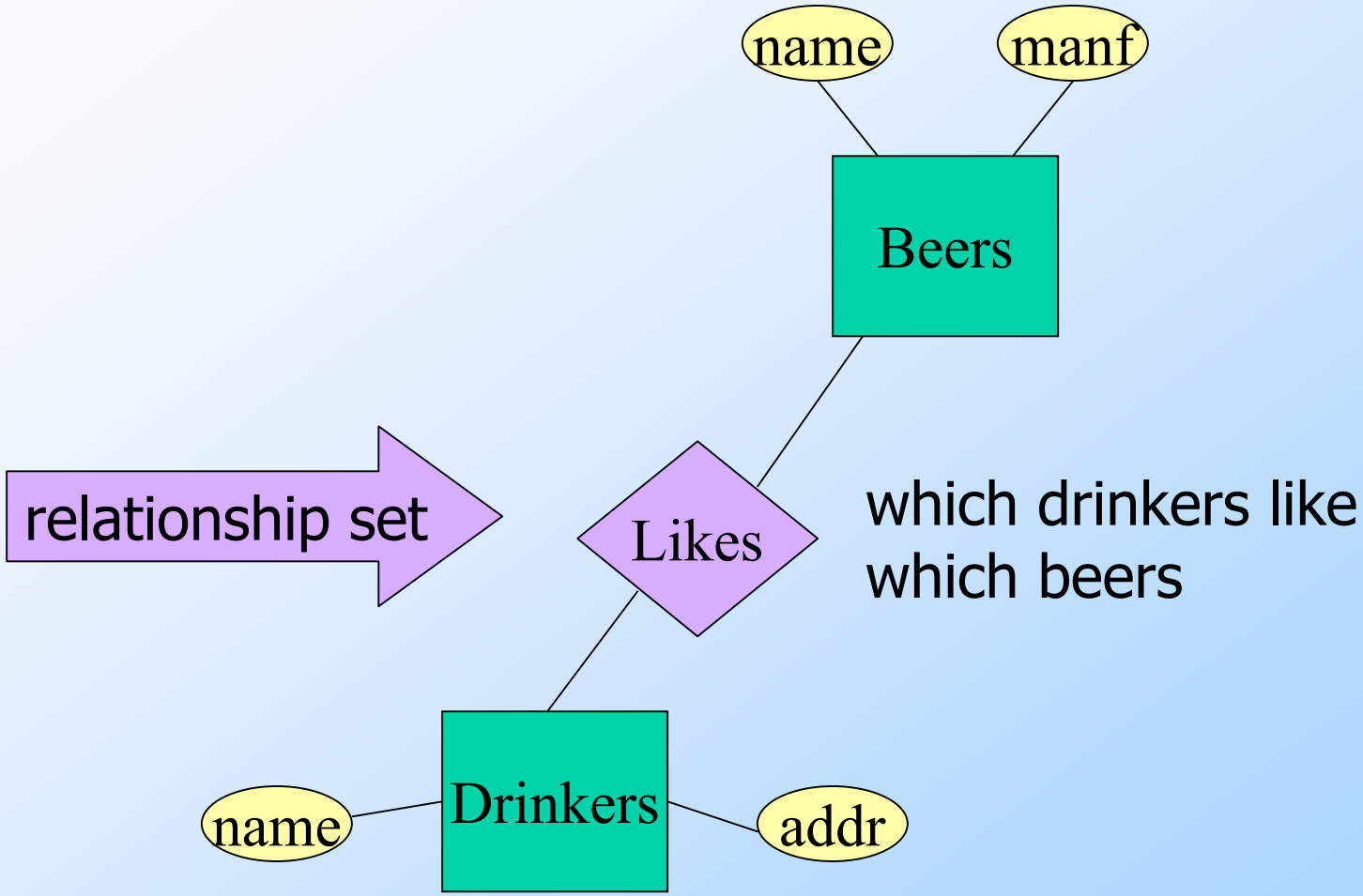
- Entity set Beers has two attributes, name and manf (manufacturer).
- Each Beer entity has values for these two attributes, e.g. (Bud, Anheuser-Busch)

Relationships

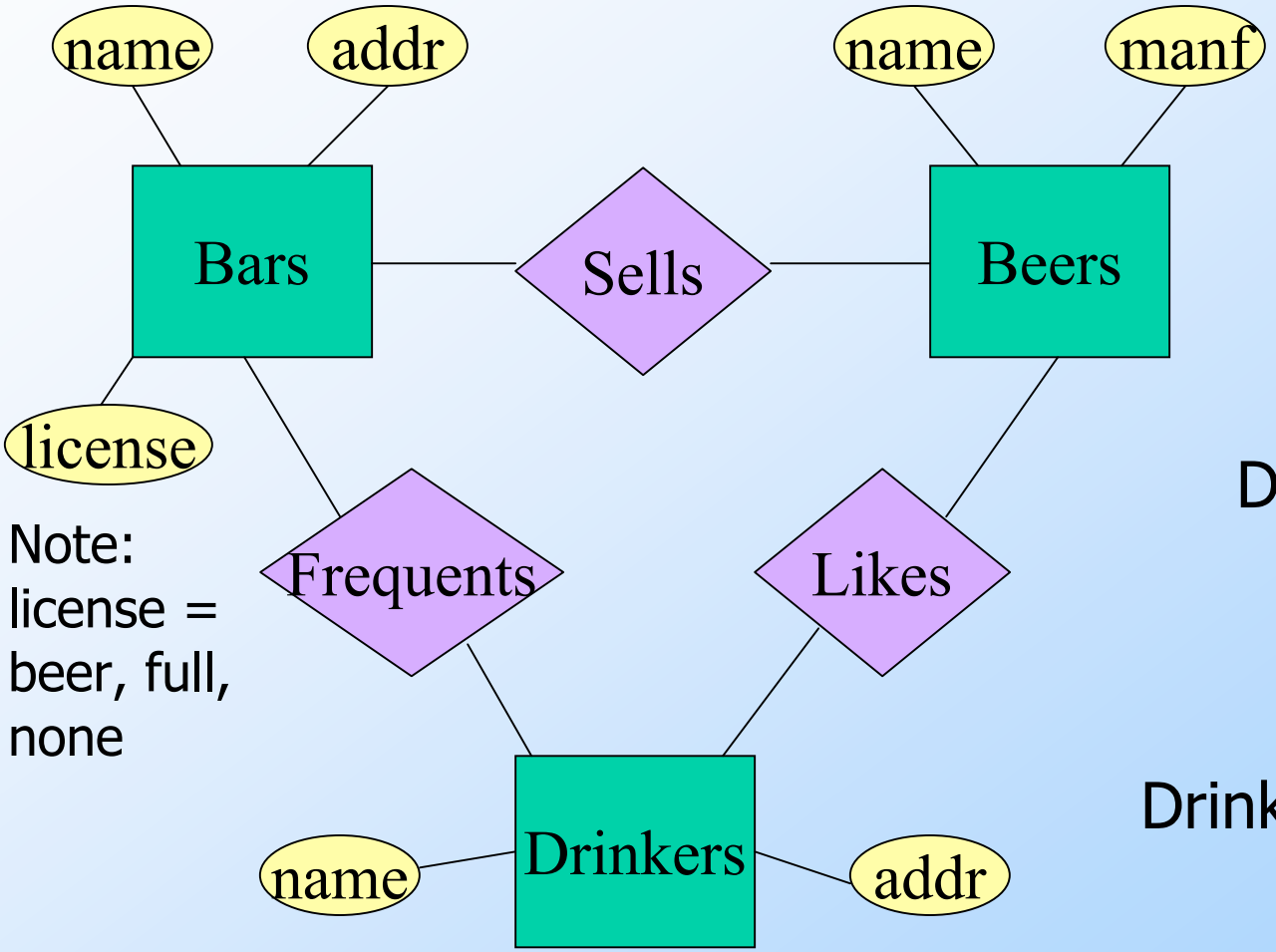
- A **relationship set** (also called "relationship") connects two or more entity sets.
  - Similar to an **association** in UML.

- A relationship set is represented by a **diamond**, with lines to each of the entity sets that are involved.

Example


name
manf
Beers
relationship set
Likes
which drinkers like
which beers
Drinkers
name
addr

New Entity and Relationship Sets



Note: license = beer, full, none

Bars sell beers.

Drinkers like beers.

Drinkers frequent 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 set is a set of tuples of currently related entities, one from each of the related entity sets.

Relationship vs. Relationship Set

- The book may use the term "relationship" when "relationship set" is intended.
- There is no good reason not to have parity in notation, especially since relationships are the more complex of relationships vs. entities.

E/R Diagram is for **Types**, not actual entities and relationships

- Regardless of the number of entities (which beers, which bars, which drinkers) and the number of relationships among those entities, it's still the same diagram.

Example

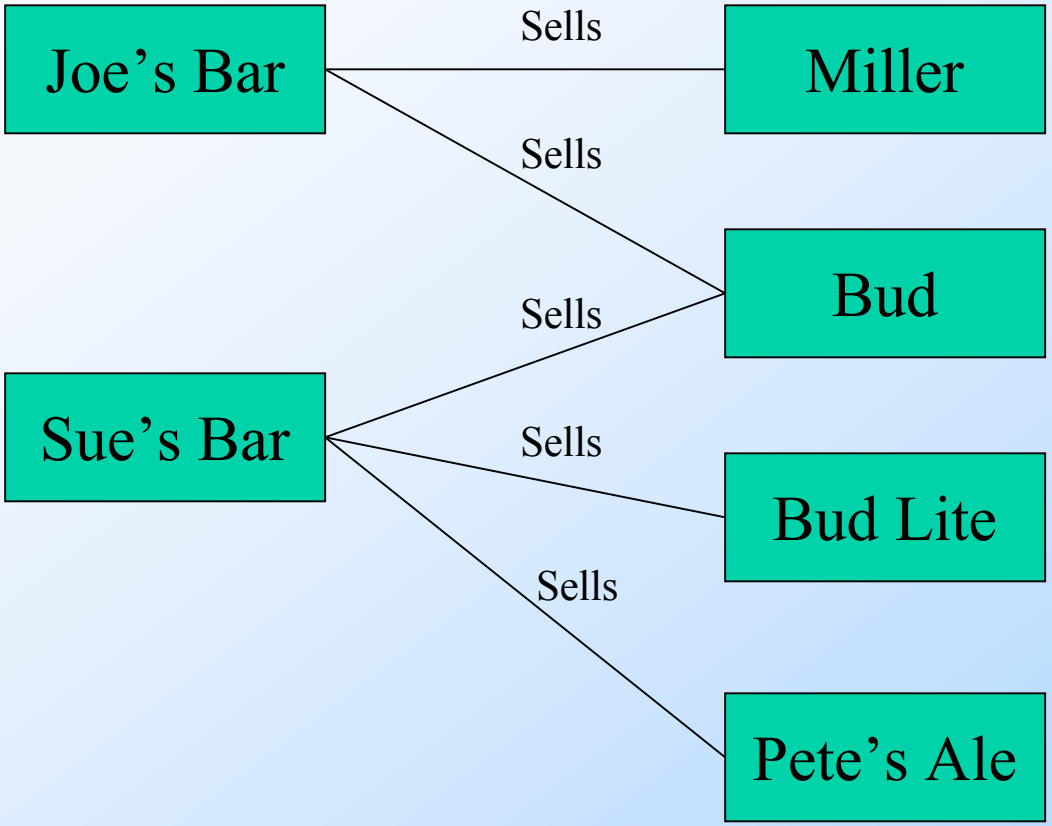
- For the relationship *Sells*, we might have a relationship set such as:

Bar	**Beer**
Joe's Bar	Bud
Joe's Bar	Miller
Sue's Bar	Bud
Sue's Bar	Pete's Ale
Sue's Bar	Bud Lite

Diagram Instantiation

- An **instance** or **instantiation** of an ER diagram has actual entities and relationships (instead of entity sets and relationship sets) as nodes.
- In AI, an instance diagram is called a "Semantic Network".

Instance Diagram for Sells

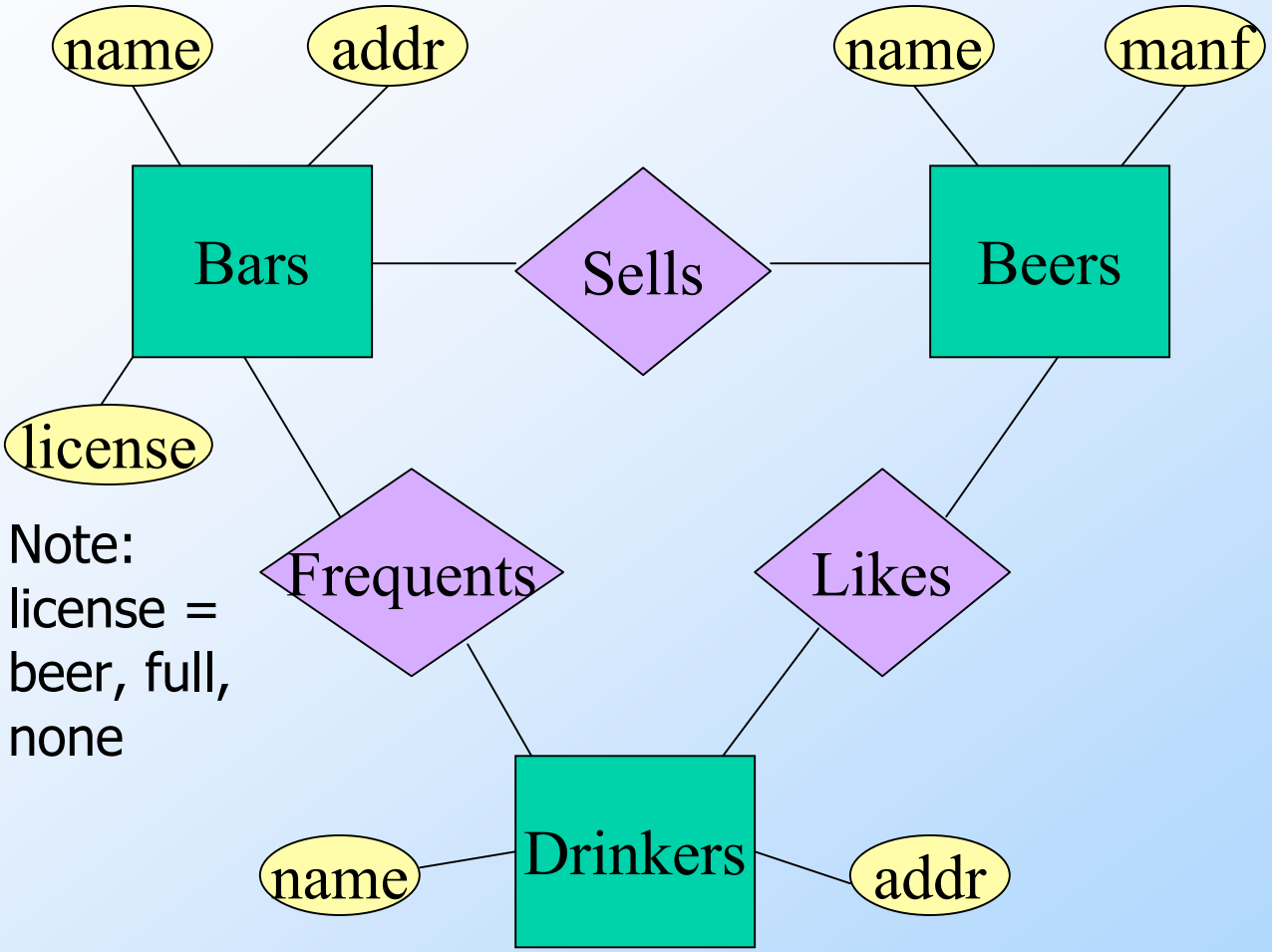


Bar	Beer
Joe's Bar	Bud
Joe's Bar	Miller
Sue's Bar	Bud
Sue's Bar	Pete's Ale
Sue's Bar	Bud Lite

Multiway Relationships

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

Two-way Relationships



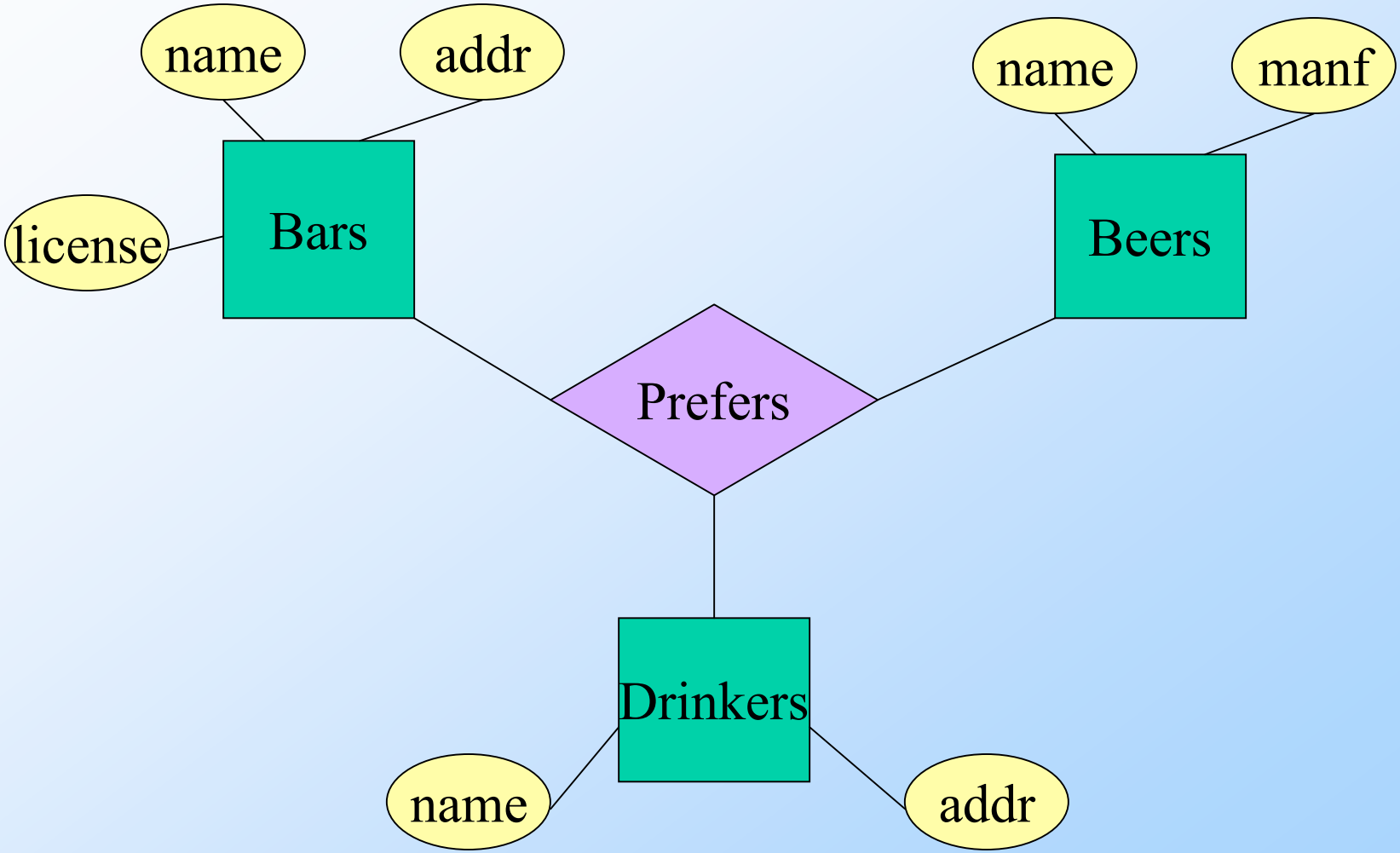
Note: license = beer, full, none

Bars sell some beers.

Drinkers like some beers.

Drinkers frequent some bars.

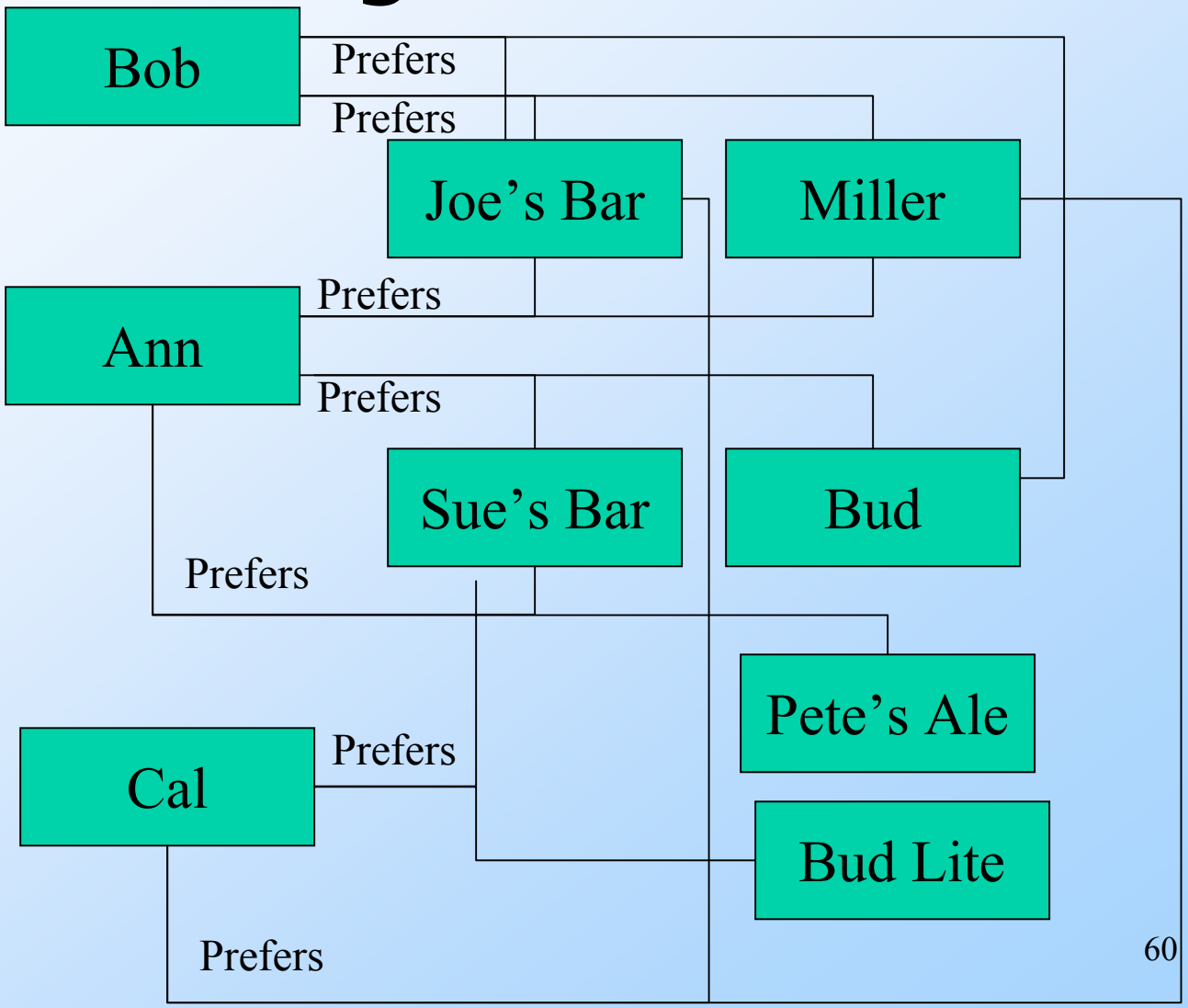
Three-Way Relationship



A Relationship Set for Prefers

Bar	Drinker	Beer
Joe's Bar	Ann	Miller
Sue's Bar	Ann	Bud
Sue's Bar	Ann	Pete's Ale
Joe's Bar	Bob	Bud
Joe's Bar	Bob	Miller
Joe's Bar	Cal	Miller
Sue's Bar	Cal	Bud Lite

Instance Diagram for Prefers



Definition: Many-One Relationships

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

Example: Many-One

- Assume that a drinker has at most one favorite beer.
- But a beer can be the favorite of any number of drinkers, including zero.
- Relationship ***Favorite***, from *Drinkers* to *Beers* is thus many-one.

Forward Comment

- Many-one relations capture the idea of a functional dependency between data in the enterprise.
- These dependencies play an important role in **normalization**, which we discuss later for the relational model.

Definition: Many-Many

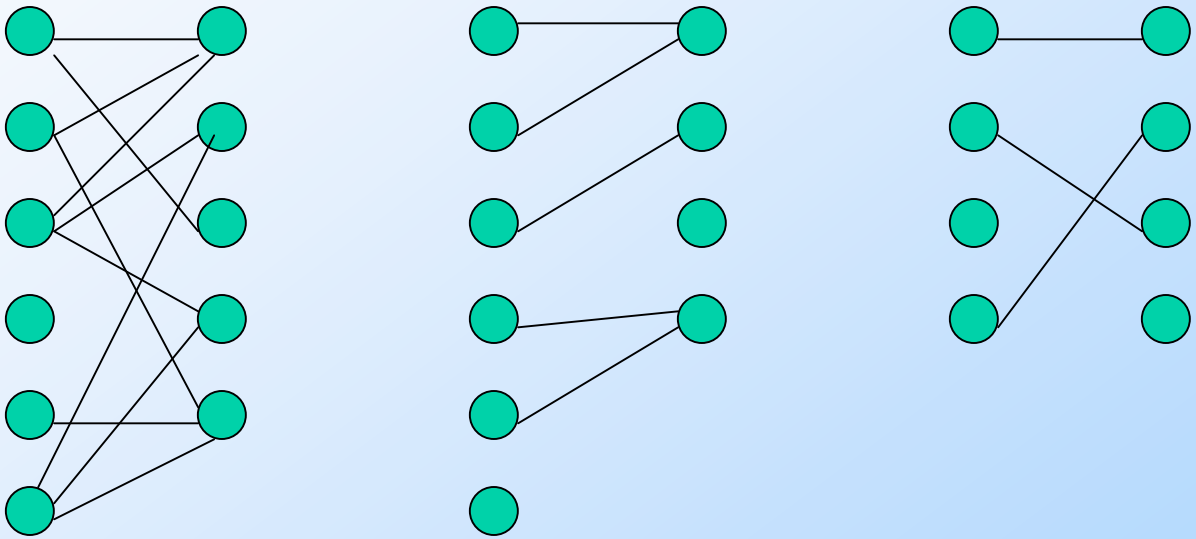
- This is the general case.
- A many-to-many relationship is not many-to-one on any dichotomy of attributes.

Definition: One-One Relationships

- In a one-one relationship between two entity sets, 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:



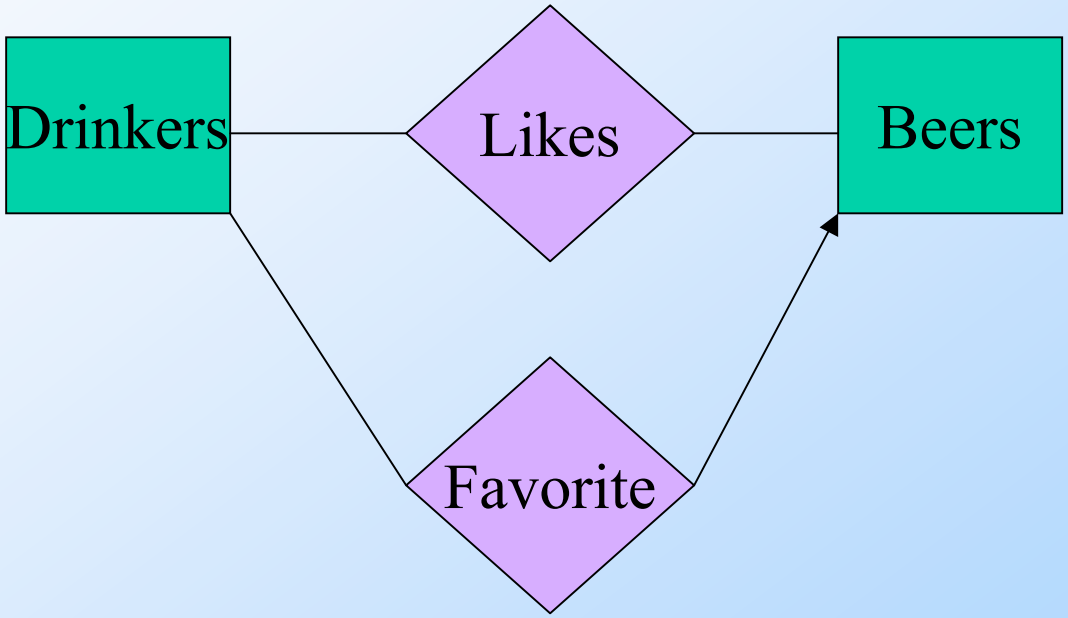
many-many many-one one-one

Note that “one-one” is not quite the same as a “one-to-one” function in mathematics.

Representing "Multiplicity"

- Show a many-one relationship by an **arrow toward** the "one" side.
- Show a one-one relationship by **bi-directional arrows**.
- In some situations, we can also assert "exactly one," i.e., each entity of one set must be related to exactly one entity of the other set. To do so, we use a **rounded arrow**.

Example



Example

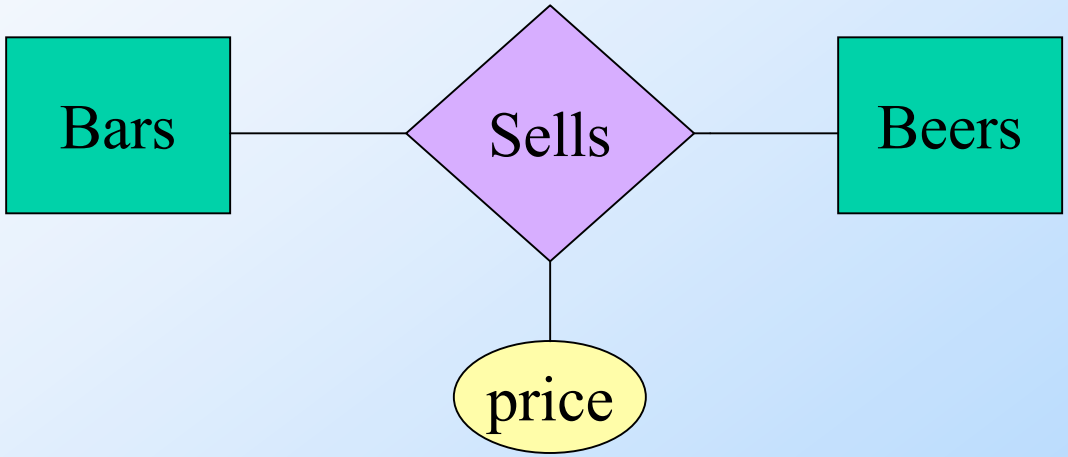
- Consider the relationship **Best-seller** between *Manfs* and *Beers*.
- Some beers are not the best-seller of any manufacturer, so a rounded arrow to *Manfs* would be inappropriate.
- But a manufacturer has to have a best-seller (we assume they are beer manufacturers).

Rounded vs. Ordinary Arrows in the E/R Diagram

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 [where "property" could be represented as another component of the tuple].

Example

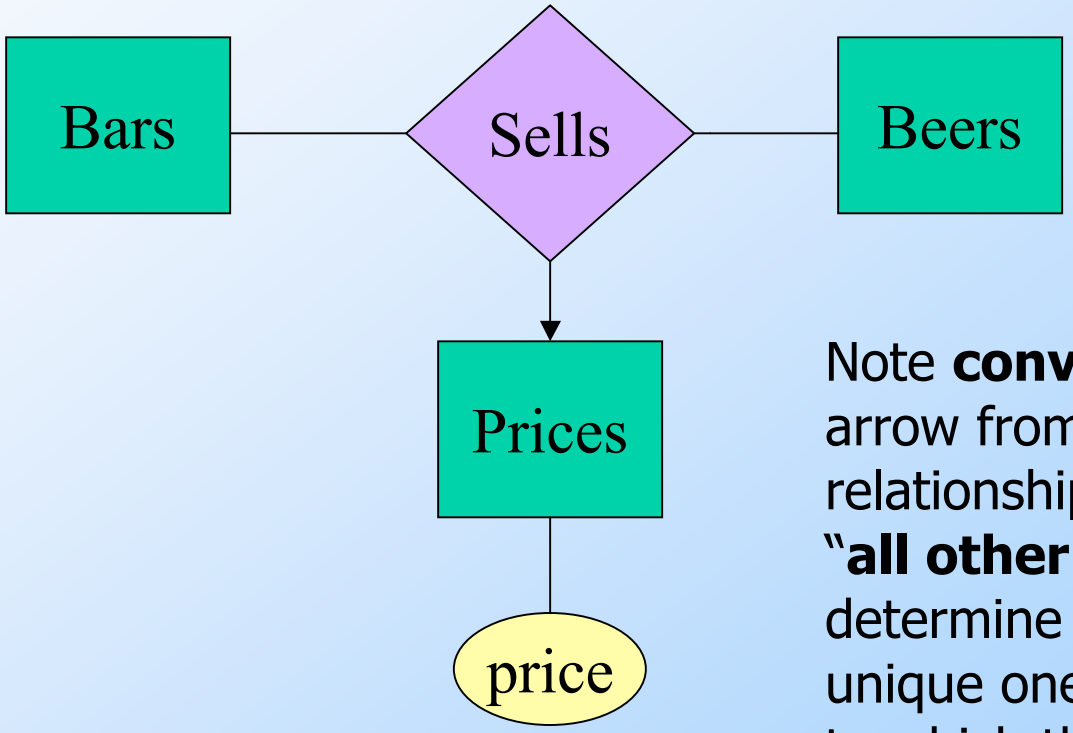


Price is a function of both the bar and the beer, not of one alone.

Equivalent Diagrams Without Attributes on Relationships

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

Example



Note **convention**:
arrow from multiway relationship means "**all other** entity sets determine a unique one of the things to which the arrow points."

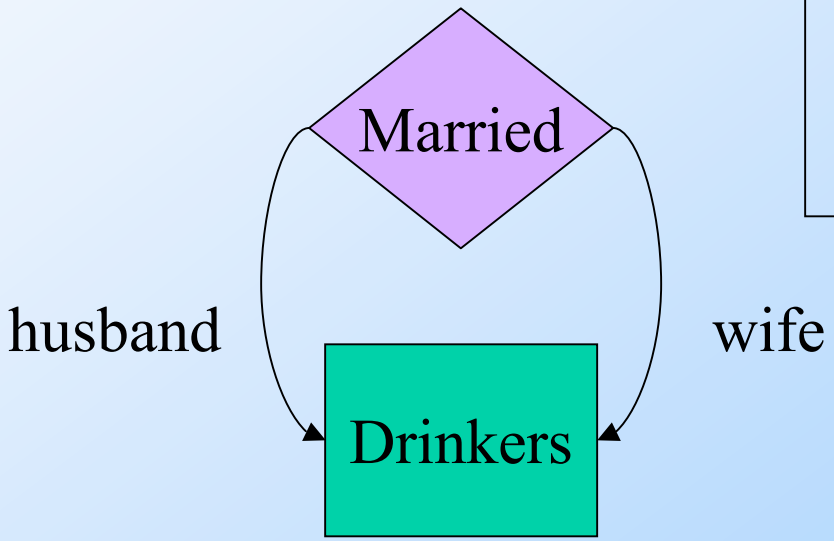
Definition:Roles

- 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*.

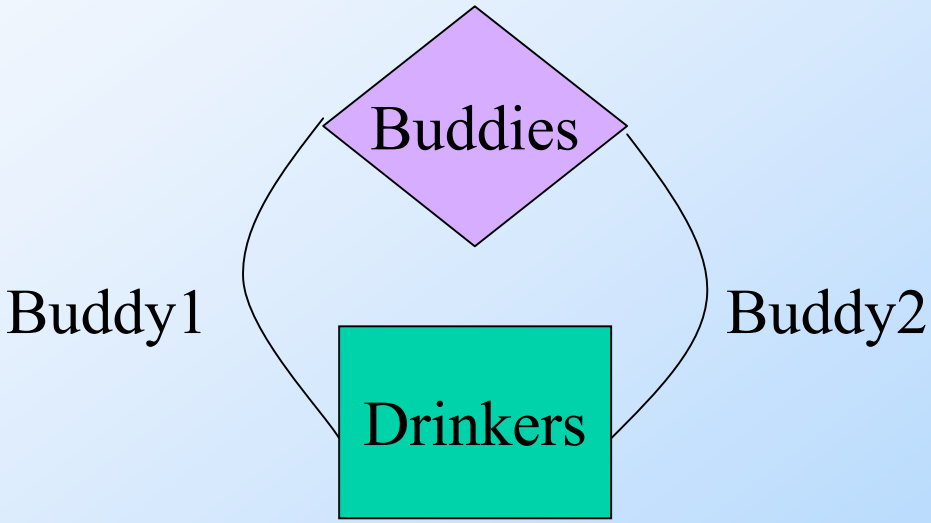
Example: Roles

Relationship Set

Husband	Wife
Bob	Ann
Joe	Sue
…	…



Example: Roles



Relationship Set

Buddy1	Buddy2
Bob	Ann
Joe	Sue
Ann	Bob
Joe	Moe
…	…

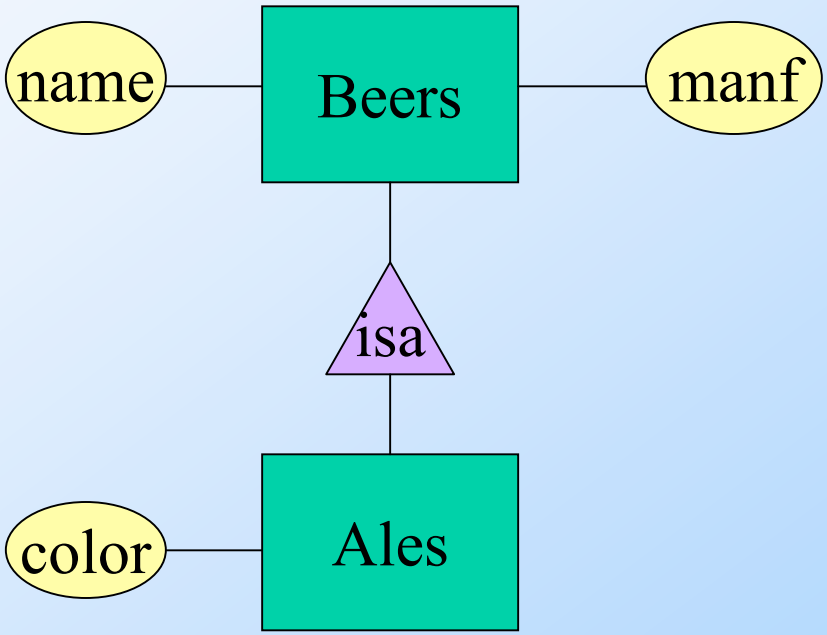
Subclasses

- 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.

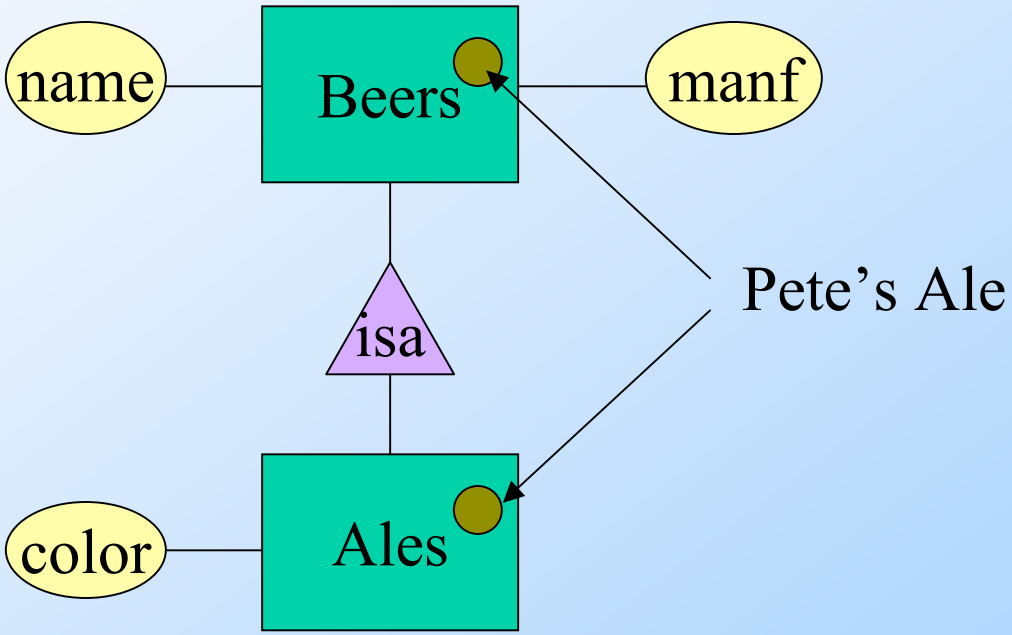
Example



E/R Vs. Object-Oriented Subclasses

- In the object-oriented world, objects are in one class only.
  - Subclasses inherit properties from superclasses.
- In contrast, E/R entities have components in **all** subclasses to which they belong.
  - This matters when we convert to relations.

Example



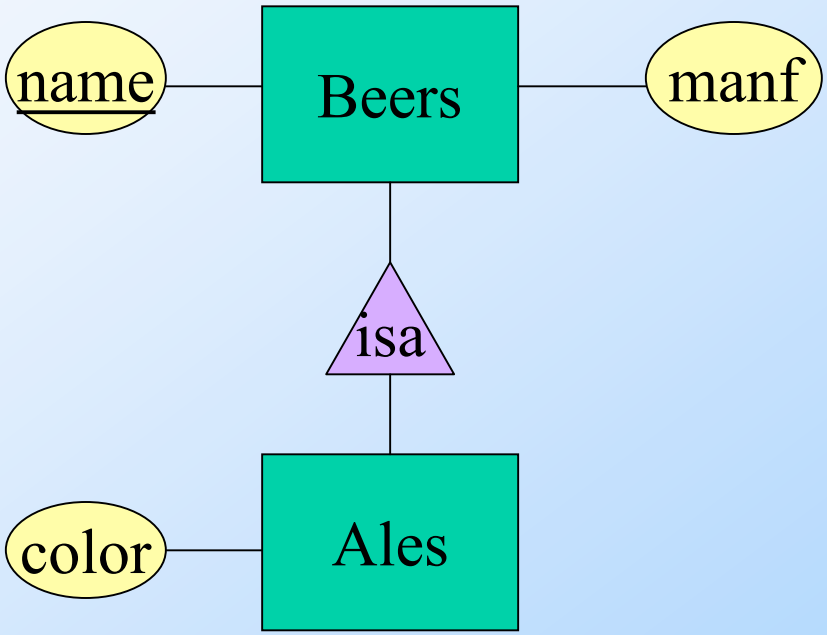
Definition: Key

- 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.

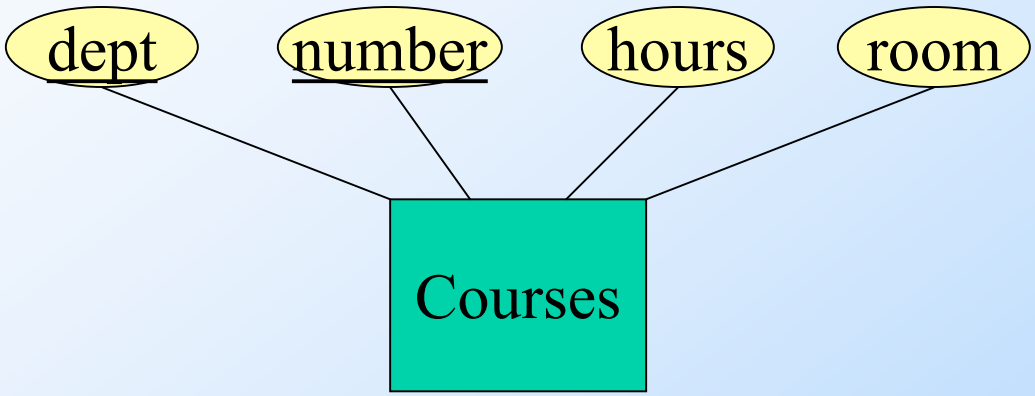
Keys in E/R Diagrams

- **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.

Definition: Weak Entity Set

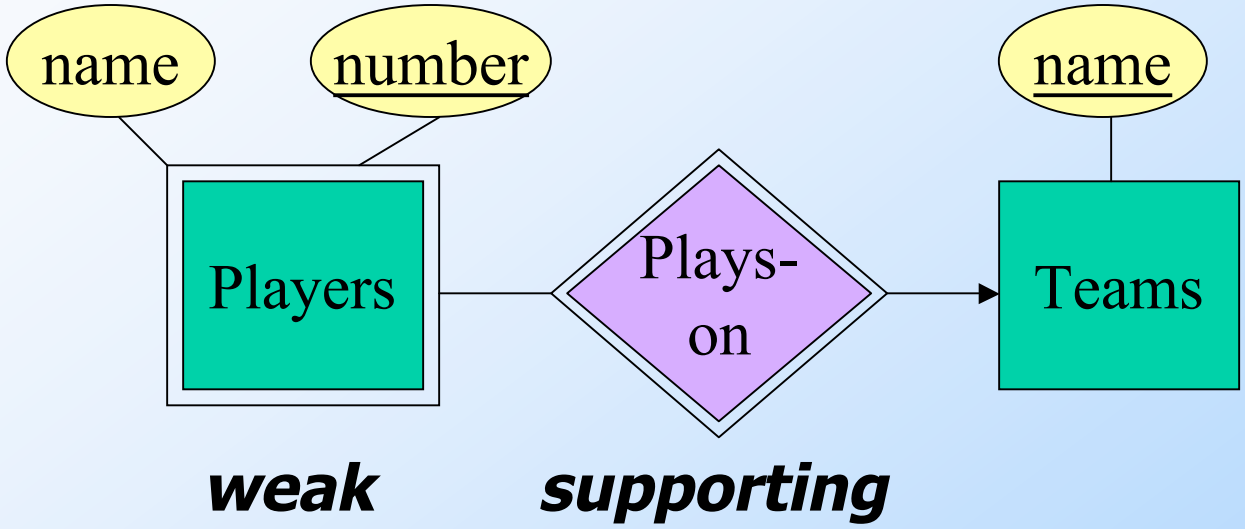
- Occasionally, entities of an entity set need "help" to identify them uniquely.

- 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 related 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*** related to the player by ***Plays-on*** should be unique.

In E/R Diagrams



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

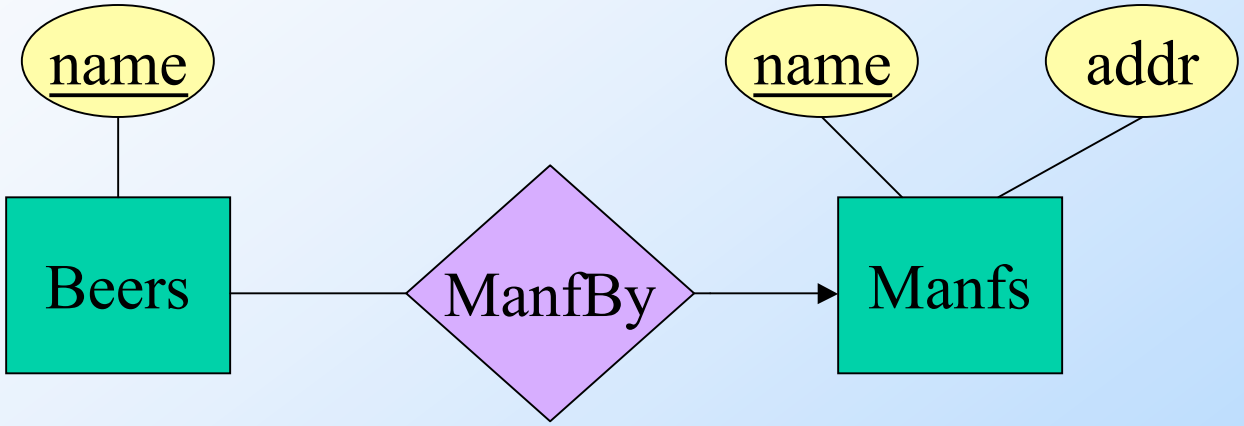
Design Guidelines

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

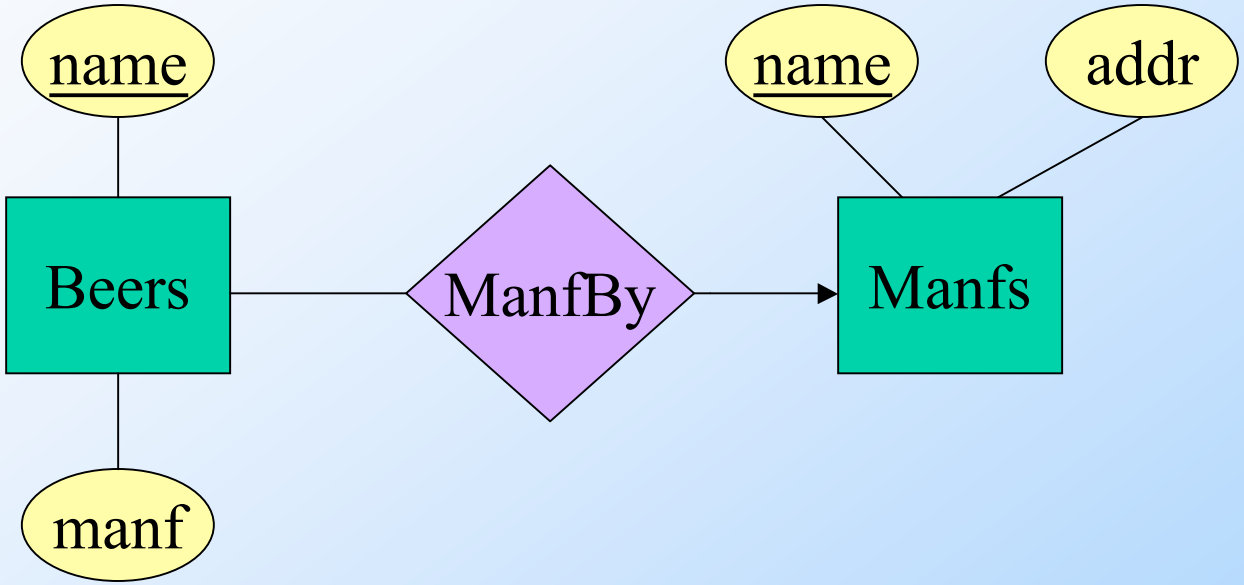
- Redundancy occurs when we say the same thing in two different ways.
- Redundancy wastes space and (more importantly) encourages inconsistency.
  - The two instances of the same fact may become inconsistent if we change one and forget to change the other, related version.

Example: Good



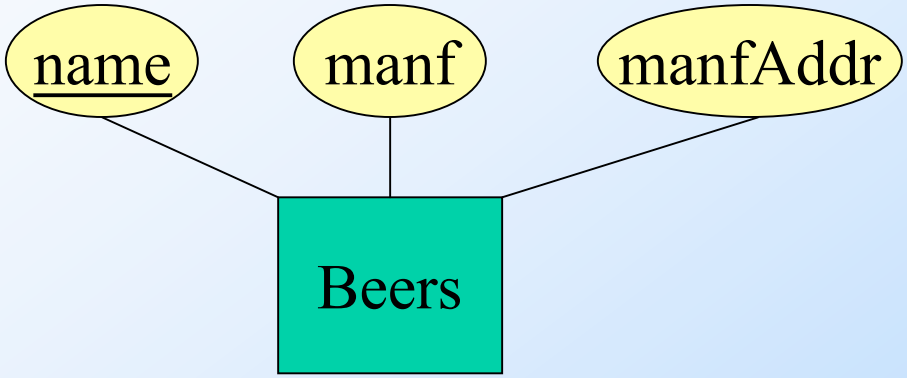
This design gives the address of each manufacturer exactly once.

Example: Bad



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

Example: Bad



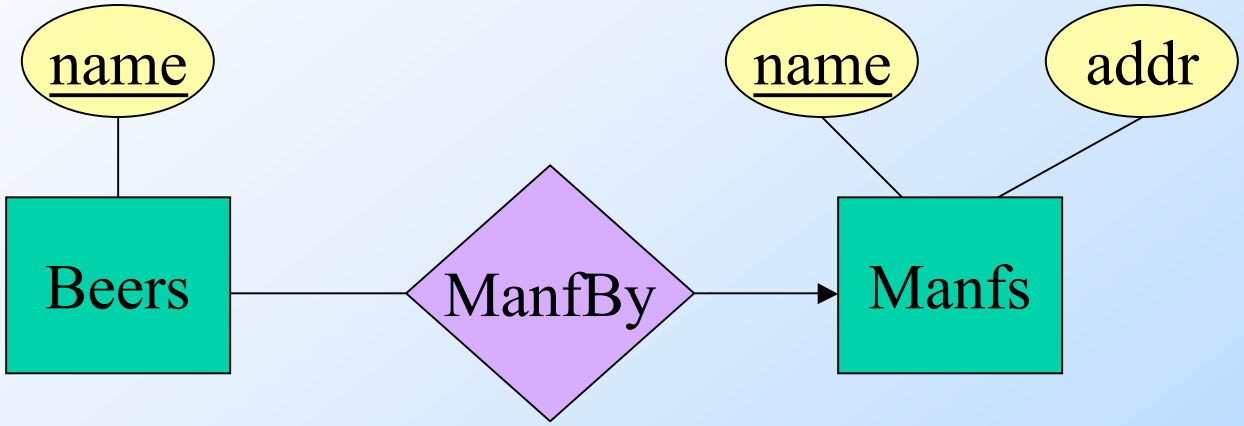
This design repeats the manufacturer's address once for each beer; loses the address if there are temporarily no beers for a manufacturer.

Entity Sets Versus Attributes

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

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

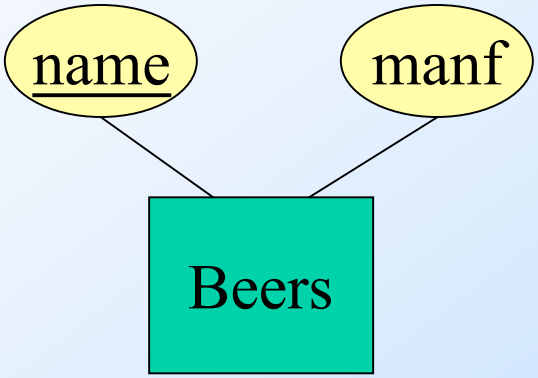
Example: Good



- *Manfs* deserves to be an entity set because of the nonkey attribute *addr*.

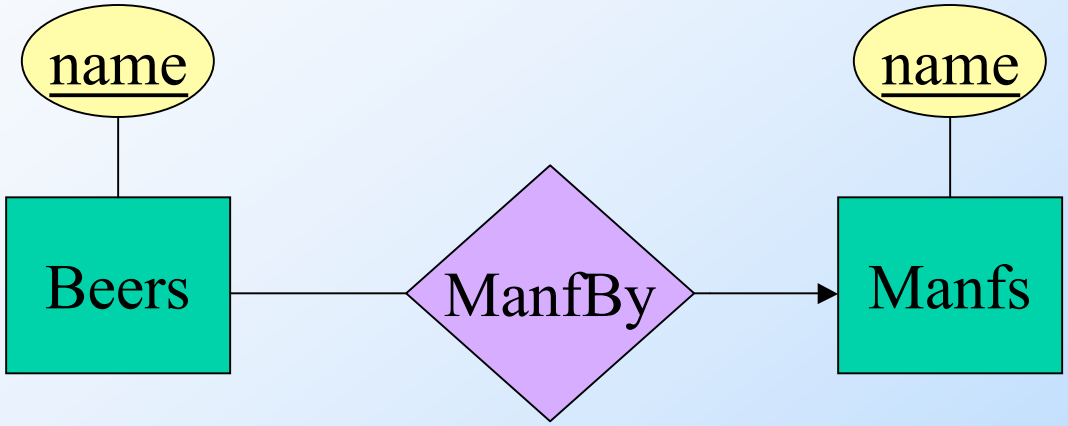
- *Beers* deserves to be an entity set because it is the "many" of the many-one relationship *ManfBy*.

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.

Exception to the Previous Rule [by RK]

- We may want to keep something as an entity even if it only has one attribute when:
  - The something **must stay represented in the database** even when not a current attribute value of any other entity.
    - For example, to offer a menu of choices.
  - To allow for evolution: We may plan to add more attributes later.

Don’t Overuse Weak Entity Sets

- Beginning database designers often overlook that an entity 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 often create unique ID’s for entities and these serve as keys.
  - Examples include social-security numbers, automobile VIN’s 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.

Additional Design Guidelines

- We will be able to propound additional guidelines within the context of the more-precise relational model.
- Many of these are expressible using the concept of functional dependency and normal forms.