Chapter 4 High-level Database Models

Entity/Relationship Models (E/R diagram)
Unified Modeling Language (UML)
Object Definition Language (ODL)
Hot to Transfer them to a relational model
Introduction

- 现实世界：客观存在的世界。
- 信息世界：现实世界在人们头脑中的反映。
- 机器世界：信息世界的在机器世界中以数据的形式存放。

<table>
<thead>
<tr>
<th>reality- » information world- » machine world</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-R data model</td>
</tr>
<tr>
<td>UML</td>
</tr>
<tr>
<td>ODL</td>
</tr>
</tbody>
</table>
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/Relationship Model

- Entity like objects, = things
- Entity set like class = set of similar Entity or objects
- Attribute = property of entities in an entity set, similar to fields of a struct.
- Relation = connect two or more entity set

In diagrams,
- entity set : rectangle;
- attribute: oval,
- relation: diamonds
Entity/Relationship Diagrams: example

- ID
- name
- address

 Students

- taking

 Course

- No.
- name
- teacher
Relationships

- **Binary** (relation between two entity sets)
- **Multiway** (relation between more than two entity sets)
- **Multiplicity** of relationships: Express the number of entities to which another entity can be associated via a relationship set.
Binary & Multiway Relationships

Students

- ID
- name
- address

Course

- taking
- No.
- name
- teacher

Students

TA

Enrolls

Course

Assisting

TA
Beers-Bars-Drinkers Example

- Bars:
  - name
  - address
  - license
  - Serves
  - Frequents

- Beers:
  - name
  - manf

- Drinkers:
  - name
  - addr

- Likes
Example: 3-Way Relationship

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.
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.
Multiplicity of Relationships

- **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.
- **Many-one** relationship: 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.
- **One-one** relationship: Each entity of either entity set is related to at most one entity of the other set.
Representing “Multiplicity”

- Show a many-one relationship by an arrow entering the “one” side.
- 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

A beer is the best-seller for 0 or 1 manufacturer.  
A manufacturer has exactly one best seller.
Attributes on Relationships

Price depends jointly on bar and beer

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

If price depends only on beers, what should we do?

Note convention:
arrow from multiway relationship = “all other entity sets together determine a unique one of these.”
Converting Multiway to 2 way

- Creating a new connecting E.S. to represent the rows of a relationship set
- Many-one relationships from the connecting E.S. to the others
Converting Multiway to 2 way: example

For each relationship \((\text{bar}_i, \text{price}_i, \text{beer}_i)\) in Sells, create:

1. a new entity \(e_i\) in the entity set \(E\)
2. add \((e_i, \text{bar}_i)\) to TheBar
3. add \((e_i, \text{price}_i)\) to ThePrice
4. add \((e_i, \text{beer}_i)\) to TheBeer

<table>
<thead>
<tr>
<th>Bars</th>
<th>Beers</th>
<th>Prices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe’s bar</td>
<td>A. B.</td>
<td>5.0</td>
</tr>
<tr>
<td>Joe’s bar</td>
<td>Bud</td>
<td>4.0</td>
</tr>
<tr>
<td>Mary’s bar</td>
<td>Bud</td>
<td>3.0</td>
</tr>
<tr>
<td>Mary’s bar</td>
<td>A. B.</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Roles

☐ Sometimes an E.S. participates more than once in a relationship.

☐ Label edges with roles to distinguish.
Subclasses

- Subclass = special case = fewer entities = more properties

- Example

  Ales are a kind of beer. In addition to the properties (= attributes and relationships) of beers, there is a “color” attribute for ales.
E/R Subclasses

- Assume subclasses form a tree (no multiple inheritance)
- Isa triangles indicate the subclass relation.

Diagram:
- beers
  - name
  - Manf.
  - isa
  - Ales
    - color
    - Point to the superclass
Multiple Inheritance

- Theoretically, an E.S. could be a subclass of several other entity sets.

- Problems?

Example: `manf` means vintner for wines, bottler for beers. What does `manf` mean for “grape beers”?

- In practice, we shall assume a tree of entity sets connected by `isa`, with all “isas” pointing from child to parent.
Different Subclass Viewpoints

- **E/R viewpoint**: 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. Its properties are the union of the properties of these E.S.

- **Object-oriented viewpoint**: An object (entity) belongs to exactly one class. It inherits properties of its superclasses.
Example, from E/R Viewpoint

- An entity has a component in each entity set to which it logically belongs.

Its properties are the union of the properties of these E.S.
Keys

A key is a set of attributes such that no two entities agree on all these attributes.

- In E/R model, every E.S. must have a key. It could have more than one key, but one set of attributes is the “designated” key.
- In E/R diagrams, you should underline all attributes of the designated key.
Example

Suppose name is key for Beers

Beer name is also key for Ales. In general, key at root is key for all.
Example: A Multiattribute Key

Possibly, hours+room also forms a key, but we have not designed it as such.
Weak Entity Sets

Sometimes an E.S. E’s key comes not (completely) from its own attributes, but from the keys of one or more E.S’s to which E is linked by a supporting many-one relationship.

- Called a weak E.S.
- Represented by putting double rectangle around E and a double diamond around each supporting relationship.
- Many-one-ness of supporting relationship (includes 1-1) essential. With many-many, we would not know which entity provided the key value.
- “Exactly one” also essential, or else we might not be able to extract key attributes by following the supporting relationship.
Example of Weak Entity Sets

- **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

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

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

(players) number and (teams) name is a key for Players

• Double diamond for supporting many-one relationship.
• Double rectangle for the weak entity set.
Example: Logins (Email addresses)

Login name = user name + host name, e.g. li-fang@cs.sjtu.edu.cn

Key for a login = the user name + Host name
Example: Logins (Email addresses)

- A “Login” entity corresponds to a user name on a particular host, but the password table does not record the host, just the user name, e.g. li-fang

- Key for a login = the user name at the host (which is unique for that host only) + IP address of the host (which is unique globally).
Example: Chain of “Weakness”: consider IP addresses consisting of a primary domain, subdomain and host.

- Key for primary domain = its name (e.g., sjtu)
- Key for secondary domain = its name (e.g., cs) + name of primary domain
- Key for host = its name (e.g., Elis) + name of 2nd Domain + name of primary domain
Relationship To Weak Entities

- Consider a relationship, Ordered, between two entity sets, Buyer and Product.

- How can we add Shipments to the mix?

  is not reasonable. Why?
Classroom Discussion

☐ What is the solution to the problem?
• Solution: make **Ordered** into a weak entity set.

• And then add **Shipment**.
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: Which one is better?

1: good

2: repeats manufacturer address for each beer they manufacture.

3: manufacturer’s name said twice.
Entity Sets Vs. Attributes

You may be unsure which concepts are worthy of being entity sets, and which are handled more simple as attributes.

Wrong ??

Right !!
Intuitive Rule for E.S.Vs. Attribute

Make an entity set only if it either:

1. Is more than a name of something; i.e., it has nonkey attributes or relationships with a number of different entity sets, or

2. Is the “many” in a many-one relationship.
Example

- Manf. Deserves to be an E.S. because we record addr, nonkey attribute.
- Beers deserves to be an E.S. because it is at the end of the “many” end.
Don’t overuse Weak E.S.

- There is a tendency to feel that no E.S. has its entities uniquely determined without following some relationships.

- However, in practice, we almost always create unique ID’s to compensate: social-security numbers, VIN’s, etc.
Don’t overuse Weak E.S.

- The only times weak E.S.’s seem necessary are when:
  1. We can not easily create such ID’s; e.g., no one is going to accept a “species ID” as part of the standard nomenclature (species is a weak E.S supported by membership in a genus)
  2. There is no global authority to create them, e.g., crews and studios.
From the view of designing: the Modeling of Constraints

- Key constraints
- Single-value constraints
- Referential integrity constraints
- Domain constraints
- General constraints

Constraints are part of the schema.
Key constraints

- No two entities may agree in their values for all of the attributes that constitute a key.
- A key may consist of more than one attribute.
- There can also be more than one possible key for an entity set.
Single-value constraints

- Many ways to express:
  - Each attribute of an entity set has a single value. (not null)
  - A relationship $R$ that is many-one from entity set $E$ to entity set $F$ implies a single-value constraint. (at most one)
Referential integrity constraints

- Every president must have a studio.
- Every studio has at most one president, that means sometimes, no president.
- If a studio (e.g. A) does not exist, its president (P1) should also be deleted as well.
Other constraints

- Domain constraints restrict the value of an attribute to be in a limited set.
- General constraints, such as placing a constraint on the degree of a relationship, number constraints and so on.
Summary of Symbols Used in E-R Notation

- E: Entity Set
- E: Weak Entity Set
- R: Relationship Set
- R: Identifying Relationship Set for Weak Entity Set
- A: Attribute
- R: Many to Many Relationship
- R: One to One Relationship
- R: Many to One Relationship
- E: Role Indicator
- ISA (Specialization or Generalization)
Exercises for E-R diagram

- Let us design a database for a bank, including information about customers and their accounts. Information about a customer includes their name, address, phone, and Social Security number. Accounts have numbers, types (e.g., savings, checking) and balances. We also need to record the customer who owns an account. Be sure to include arrows where appropriate, to indicate the multiplicity of a relationship.
E-R Diagram

customers
- name
- address
- phone
- ssNo

accounts
- type
- balance
- number

owns
Some modifications on the example

- Suppose an account can have only one customer.
- A customer can have a set of addresses (which is street-city-state triples) and a set of phones.

What are the ER diagram?
Suppose an account can have only one customer.
A customer can have a set of addresses (which is street-city-state triples) and a set of phones.

We do not allow attributes to have collection types in the E/R diagram.
Question?

- Modify the diagram so that customers can have a set of addresses, and at each address there is a set of phones.
Classroom Design Exercise

- Design a database suitable for a university registrar. This database should include information about students, departments, professors, courses, which students are enrolled in which courses, which professors are teaching which courses, students grades, TA’s for a course (TA’s are students), which courses a department offers, and any other information you deem appropriate.
Summary of E/R diagram

- Entity-Relationship Diagrams
  - Entities & Attributes & Relationships
  - Multiplicity, Multiway of Relationships
  - Weak Entity Sets
  - Subclasses
- Good Design
  - Faithfully represent
  - Avoid redundancy
  - Choose appropriate elements
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

Drinkers

Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Beers

name
addr
name
manf

husband
wife

Buddies

Married

1
2
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).
Combining *Drinkers* with *Likes* would be a mistake. It leads to redundancy, as:

```
<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>123 Maple</td>
<td>Bud</td>
</tr>
<tr>
<td>Sally</td>
<td>123 Maple</td>
<td>Miller</td>
</tr>
</tbody>
</table>
```

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, billTo)
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

![Diagram showing subclass relationships between Beers and Ales]

- **Beers** has attributes: name, manf, and isa relationship with **Ales**.
- **Ales** has attribute: color.
Object-Oriented

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>Anheuser-Busch</td>
</tr>
</tbody>
</table>

Beers

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summerbrew</td>
<td>Pete’s</td>
<td>dark</td>
</tr>
</tbody>
</table>

Ales

Good for queries like “find the color of ales made by Pete’s.”
E/R Style

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>Anheuser-Busch</td>
</tr>
<tr>
<td>Summerbrew</td>
<td>Pete’s</td>
</tr>
</tbody>
</table>

Beers

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summerbrew</td>
<td>dark</td>
</tr>
</tbody>
</table>

Ales

Good for queries like
“find all beers (including ales) made by Pete’s.”
Using Nulls

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>Anheuser-Busch</td>
<td>NULL</td>
</tr>
<tr>
<td>Summerbrew</td>
<td>Pete’s</td>
<td>dark</td>
</tr>
</tbody>
</table>

Beers

Saves space unless there are *lots* of attributes that are usually NULL.
Other High-Level Design Languages

Object Definition Language (ODL)
Unified Modeling Language (UML)
Object-Oriented DBMS’s

- Standards group: ODMG = Object Data Management Group.
- ODL = Object Description Language, like CREATE TABLE part of SQL.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.
ODMG imagines OO-DBMS vendors implementing an OO language like C++ with extensions (OQL) that allow the programmer to transfer data between the database and “host language” seamlessly.
Framework - (2)

- ODL is used to define *persistent* classes, whose objects are stored permanently in the database.
  - ODL classes look like Entity sets with binary relationships, plus methods.
  - ODL class definitions are part of the extended, OO host language.
ODL Overview

- A class declaration includes:
  1. A name for the class.
  2. Optional key declaration(s).
  3. Element declarations. An element is either an attribute, a relationship, or a method.
Class Definitions

class <name> {  
    <list of element declarations, separated by semicolons>
}
Attribute and Relationship Declarations

- Attributes are (usually) elements with a type that does not involve classes.
  
  \[ \text{attribute} \ <\text{type}> \ <\text{name}>; \]

- Relationships connect an object to one or more other objects of one class.
  
  \[ \text{relationship} \ <\text{type}> \ <\text{name}> \ \text{inverse} \ <\text{relationship}>; \]
Inverse Relationships

- Suppose class $C$ has a relationship $R$ to class $D$.
- Then class $D$ must have some relationship $S$ to class $C$.
- $R$ and $S$ must be true inverses.
  
  - If object $d$ is related to object $c$ by $R$, then $c$ must be related to $d$ by $S$. 
Example: Attributes and Relationships

```java
class Bar {
    attribute string name;
    attribute string addr;
    relationship Set<Beer> serves inverse Beer::servedAt;
}

class Beer {
    attribute string name;
    attribute string manf;
    relationship Set<Bar> servedAt inverse Bar::serves;
}
```

The type of relationship serves is a set of Beer objects.
The :: operator connects a name on the right to the context containing that name, on the left.
Types of Relationships

- The type of a relationship is either
  1. A class, like Bar. If so, an object with this relationship can be connected to only one Bar object.
  2. Set<Bar>: the object is connected to a set of Bar objects.
  3. Bag<Bar>, List<Bar>, Array<Bar>: the object is connected to a bag, list, or array of Bar objects.
Multiplicity of Relationships

☐ All ODL relationships are binary.

☐ Many-many relationships have Set<...> for the type of the relationship and its inverse.

☐ Many-one relationships have Set<...> in the relationship of the “one” and just the class for the relationship of the “many.”

☐ One-one relationships have classes as the type in both directions.
Example: Multiplicity

```cpp
class Drinker {
    relationship Set<Beer> likes inverse Beer::fans;
    relationship Beer favBeer inverse Beer::superfans;
}

class Beer {
    relationship Set<Drinker> fans inverse Drinker::likes;
    relationship Set<Drinker> superfans inverse Drinker::favBeer;
}
```

Many-many uses `Set<..>` in both directions.
Many-one uses `Set<..>` only with the “one.”
Another Multiplicity Example

class Drinker {
    attribute ...;
    relationship Drinker husband inverse wife;
    relationship Drinker wife inverse husband;
    relationship Set<Drinker> buddies inverse buddies;
}

husband and wife are one-one and inverses of each other.

buddies is many-many and its own inverse. Note no :: needed if the inverse is in the same class.
Coping With Multiway Relationships

- ODL does not support 3-way or higher relationships.
- We may simulate multiway relationships by a “connecting” class, whose objects represent tuples of objects we would like to connect by the multiway relationship.
Connecting Classes

- Suppose we want to connect classes $X$, $Y$, and $Z$ by a relationship $R$.
- Devise a class $C$, whose objects represent a triple of objects $(x, y, z)$ from classes $X$, $Y$, and $Z$, respectively.
- We need three many-one relationships from $(x, y, z)$ to each of $x$, $y$, and $z$. 
Example: Connecting Class

- Suppose we have Bar and Beer classes, and we want to represent the price at which each Bar sells each beer.
  - A many-many relationship between Bar and Beer cannot have a price attribute as it did in the E/R model.
- One solution: create class Price and a connecting class BBP to represent a related bar, beer, and price.
Example -- Continued

Since Price objects are just numbers, a better solution is to:

1. Give BBP objects an attribute price.
2. Use two many-one relationships between a BBP object and the Bar and Beer objects it represents.
Example -- Concluded

- Here is the definition of BBP:
  
  ```
  class BBP {
      attribute price:real;
      relationship Bar theBar inverse Bar::toBBP;
      relationship Beer theBeer inverse Beer::toBBP;
  }
  ```

- Bar and Beer must be modified to include relationships, both called toBBP, and both of type Set<BBP>.
Structs and Enums

- Attributes can have a structure (as in C) or be an enumeration.
- Declare with
  attribute [Struct or Enum] <name of struct or enum> { <details> }
  <name of attribute>;
- Details are field names and types for a Struct, a list of constants for an Enum.
Example: Struct and Enum

class Bar {
    attribute string name;
    attribute Struct Addr{string street, string city, int zip} address;
    attribute Enum Lic { FULL, BEER, NONE } license;
    relationship ...
}

Names for the structure and enumeration

Addresses

License

Names of the attributes
Method Declarations

- A class definition may include declarations of methods for the class.
- Information consists of:
  1. Return type, if any.
  2. Method name.
  3. Argument modes and types (no names).
     - Modes are in, out, and inout.
  4. Any exceptions the method may raise.
Example: Methods

real gpa(in string) raises(noGrades);

1. The method gpa returns a real number (presumably a student’s GPA).
2. gpa takes one argument, a string (presumably the name of the student) and does not modify its argument.
3. gpa may raise the exception noGrades.
The ODL Type System

- Basic types: int, real/float, string, enumerated types, and classes.
- Type constructors:
  - Struct for structures.
  - *Collection types*: Set, Bag, List, Array, and Dictionary ( = mapping from a domain type to a range type).
- Relationship types can only be a class or a single collection type applied to a class.
ODL Subclasses

- Usual object-oriented subclasses.
- Indicate superclass with a colon and its name.
- Subclass lists only the properties unique to it.
  - Also inherits its superclass’ properties.
Example: Subclasses

- Ales are a subclass of beers:

```java
class Ale : Beer {
    attribute string color;
}
```
ODL Keys

- You can declare any number of keys for a class.
- After the class name, add:
  (key <list of keys>)
- A key consisting of more than one attribute needs additional parentheses around those attributes.
Example: Keys

class Beer (key name) { ...

name is the key for beers.

class Course (key (dept, number), (room, hours)) {

dept and number form one key; so do room and hours.
Translating ODL to Relations

☐ Classes without relationships: like entity set, but several new problems arise.

☐ Classes with relationships:
   a) Treat the relationship separately, as in E/R.
   b) Attach a many-one relationship to the relation for the “many”.
ODL Class Without Relationships

- Problem: ODL allows attribute types built from structures and collection types.

- Structure: Make one attribute for each field.

- Set: make one tuple for each member of the set. More than one set attribute? Make tuples for all combinations.

- Problem: ODL class may have no key, but we should have one in the relation to represent “OID”.
Example

Class Drinkers (key name)

{ attribute string name;
  attribute Struct Addr { string street,
    string city, int zip} address;
  attribute Set <string> phone; }

<table>
<thead>
<tr>
<th>Name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>n₁</td>
<td>s₁</td>
<td>c₁</td>
<td>z₁</td>
<td>p₁</td>
</tr>
<tr>
<td>n₁</td>
<td>s₁</td>
<td>c₁</td>
<td>z₁</td>
<td>p₂</td>
</tr>
</tbody>
</table>
Example (cont.)

- Surprise: the key for class (name) is not the key for the relation (name, phone)
- Name in the class determines a unique object, including a set of phones.
- Name in the relation does not determine a unique tuple.
- Since tuples are not identical to objects, there is no inconsistency!
- BCNF violation: separate out name-phone.
ODL Relationships

- Create for each relationship a new relation that connects the keys of the two related classes, one relation for each pair.
- If the relationship is many-one from A to B, put key of B attributes in the relation for class A.
Example

Class Drinkers (key name) {
    attribute string name;
    attribute string addr;
    relationship Set<Beers> likes inverse Beers:: fans;
    relationship Beers favorite inverse Beers:: realFans;
    Relationship Drinkers husband inverse wife;
    Relationship Drinkers wife inverse husband;
    Relationship Set<Drinkers> buddies inverse buddies;
}

Drinkers (name, addr, favBeer, marriedwith)
Likes (drinkerName, Beersname)
Buddy (drinker1,drinker2)
UML introduction

- UML is an acronym for Unified Modeling Language.

- The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting
  the artifacts of a software-intensive system.

- Object-Oriented & Visual Modeling
UML

- UML is designed to model software, but has been adapted as a database modeling language.
- Midway between E/R and ODL.
  - No multiway relationships as in E/R.
  - But allows attributes on binary relationships, which ODL doesn’t.
  - Has a graphical notation, unlike ODL.
Classes

- Sets of objects, with attributes \((\textit{state})\) and methods \((\textit{behavior})\).
- 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

Class Name: Bar

PK Name: string
Addr: string

Methods:
- setName(n)
- setAddr(a)
- getName() : string
- getAddr() : string
- sellsBud() : boolean
Associations

- Binary relationships between classes.
- Represented by named lines (no diamonds as in E/R).
- 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

<table>
<thead>
<tr>
<th>Bar</th>
<th>1..50 Sells 0..*</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

109
Comparison With E/R Multiplicities

<table>
<thead>
<tr>
<th>E/R</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Association Classes

- Attributes on associations are permitted.
  - Called an \textit{association class}.
  - Analogous to attributes on relationships in E/R.
Example: Association Class

```
<table>
<thead>
<tr>
<th>Bar</th>
<th>1..50</th>
<th>Sells</th>
<th>price: float</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Beer</th>
<th>0..*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Subclasses

- Like E/R, but 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 in UML

![UML diagram showing subclass relationships between Beer and Ale. Beer has attributes name: string and manf: string. Ale inherits from Beer and has attribute color: string.]}
Subclasses (cont.)

- In a typical object-oriented system, subclasses are disjoint.
- E/R model allows overlapping subclasses.
- E/R model and object-oriented system allow either complete or partial subclasses. There is no requirement that a member of the superclass be in any of subclass.
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.
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.
Examples of Aggregation and composition

- An aggregation from Movies to Studios (many-one relationship)
- A composition from Presidents to Studios (the label at the diamond end must be 1..1)

The implication of the composition is that presidents objects will contain a reference to a Studios object and that this reference can not be null.
Examples of Aggregation and composition (cont.)

- Both represent Part-whole relationship
- Composition has a strong part-whole relationship, the part and the whole have the same life cycle.
## Comparison between UML and E/R model

<table>
<thead>
<tr>
<th>UML</th>
<th>E/R Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Entity set</td>
</tr>
<tr>
<td>Association</td>
<td>Binary relationship</td>
</tr>
<tr>
<td>Association class</td>
<td>Attributes on a relationship</td>
</tr>
<tr>
<td>Subclass</td>
<td>Isa hierarchy</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Many-one relationship</td>
</tr>
<tr>
<td>Composition</td>
<td>Many-one relationship with referential integrity</td>
</tr>
</tbody>
</table>
Conversion to Relations

- We can use any of the three strategies outlined for E/R 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.
Conversion to Relations (cont.)

- Classes to Relations.
- Associations to Relations.

✓ Aggregations and compositions are types of many-one associations. Construct no relations for them.
From Aggregation to relations

- We could store the awards of a beer with the beer tuple.
- Requires an object-relational or nested-relation model for tables, since there is no limit to the number of awards a beer can win.
• 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.
• In object-relational databases, Addr can be one attribute of Bars, with structure.
Relationship Comparison between models

- E/R model: many-to-many relationships, *multiway relationship*, relationship can have an attribute
- UML: many-to-many relationships, relationship can have an attribute
- ODL: many-to-many relationships, *relationship has not attributes*, with inverse relationship.
Summary

- The E/R model (subclass, weak entity sets)
- UML model
- ODL (keys, relationships, type system)
- Transfer E/R to relational model (Isa hierarchies)
- Transfer UML to relations
- Transfer ODL to relations