Relational Databases

http://goo.gl/CDWTvr

CS5200 DBMS
Bruce Chhay

Relational Databases

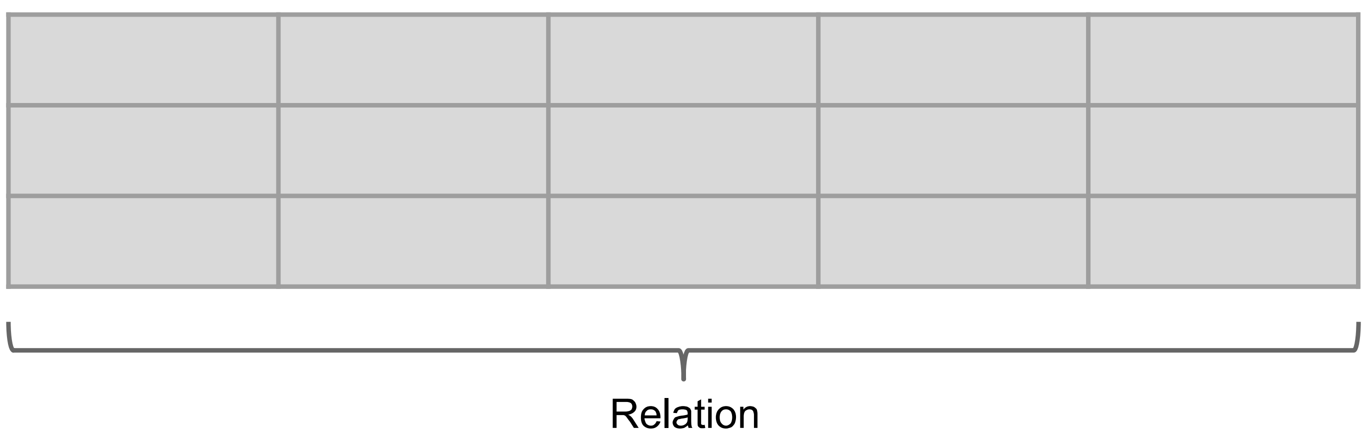
L1: Terminology

Relational Database

- A **relational database** is a collection of related tables, where each table consists of rows and columns.
- Structured Query Language (**SQL**) is a declarative language used to interact with relational data.
- Although defined in the 1970's by IBM, the theoretical foundation is **relational algebra**, which is still relevant in modern DBs!

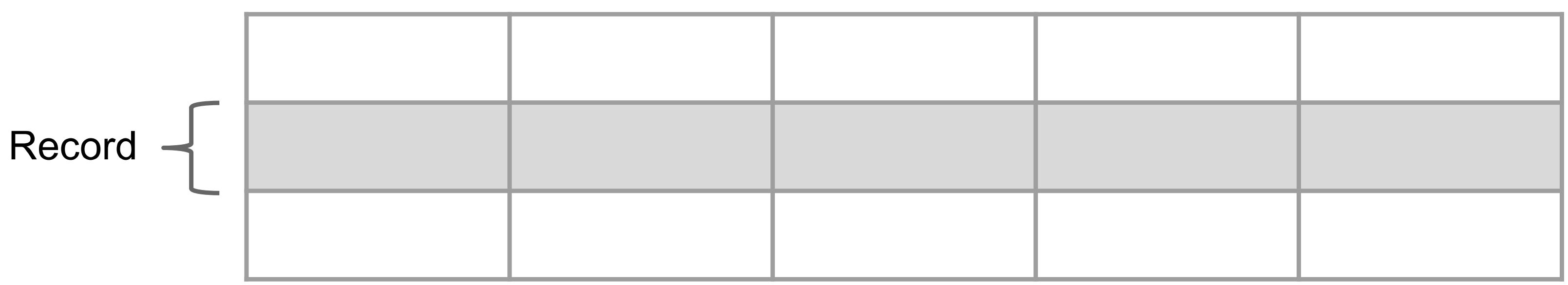
Terminology

- **Table == Relation.** A set of items with the same fields.
- **Row == Record or Tuple**. A single item.
- **Column == Attribute or Field**. Property of the relation.



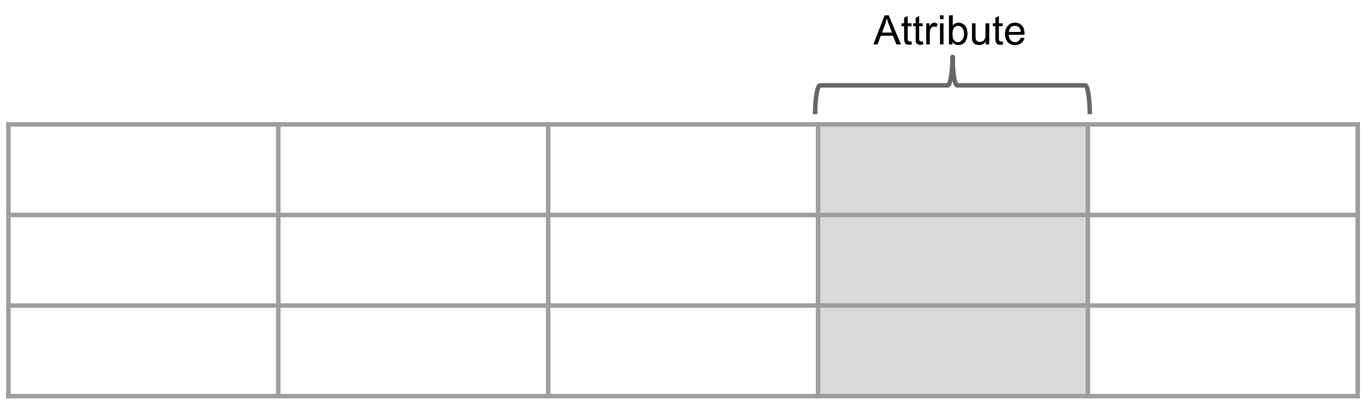
Terminology

- **Table == Relation**. A set of items with the same fields.
- **Row == Record or Tuple**. A single item.
- **Column == Attribute or Field**. Property of the relation.



Terminology

- **Table == Relation**. A set of items with the same fields.
- **Row == Record or Tuple**. A single item.
- **Column == Attribute**. Property of the relation.



Relational Database

- A **relational database** is a collection of related tables, where each table consists of rows and columns.
- Structured Query Language (**SQL**) is a declarative language used to interact with relational data.
- Although defined in the 1970's by IBM, the theoretical foundation is **relational algebra**, which is still relevant in modern DBs!

Declarative vs Control Flow

Declarative (describe what you want in the result output):

```
SELECT id
FROM students
WHERE name == ‘jae’
```

Control Flow (describe how to accomplish it, including side effects):

```
for student in students:
  if student.name == ‘jae’:
    print student.id
```

Declarative vs Control Flow

Declarative (describe what you want in the result output):

```
SELECT id
FROM students
WHERE name == ‘jae’
```

Control Flow (describe how to accomplish it, including side effects):

```
for student in students:
  if student.name == ‘jae’:
    print student.id
```

Relational Database

- A **relational database** is a collection of related tables, where each table consists of rows and columns.
- Structured Query Language (**SQL**) is a declarative language used to interact with relational data.
- Although defined in the 1970's by IBM, the theoretical foundation is **relational algebra**, which is still relevant in modern DBs!

Relational Algebra

Basic operations:

- Selection
  (restriction/subset of rows)
- Projection
  (subset of attributes)
- Set operations (combine tables)

```
SELECT id
FROM students
WHERE name == 'jae'
```

Relational Algebra

Basic operations:

- Selection (restriction/subset of rows)
- Projection (subset of attributes)
- Set operations (combine tables)

```
SELECT id
FROM students
WHERE name == 'jae'
```

Relational Algebra

Basic operations:

- Selection
  (restriction/subset of rows)
- Projection
  (subset of attributes)
- Set operations
  (combine tables)

```
SELECT id
FROM students
WHERE name == 'jae'
```

Terminology

- **Normalization**: process of reorganizing to reduce redundancy.
- **Constraints**: enforcement of record integrity, referential integrity, business rule integrity.

Terminology: Classes

- Class == Table == Relation
- Instance == Row == Record/Tuple
- Field == Column == Attribute

Relational Databases

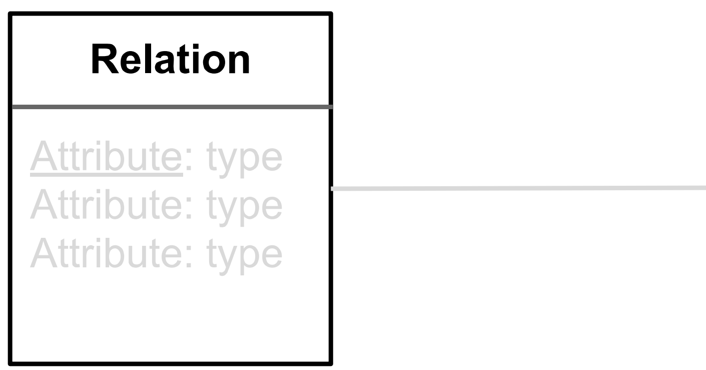
L2: UML

Communicating a Relational Model

- Use a diagram to communicate how objects are organized to form a system.
- **Unified Modeling Language (UML)** to describe relations and relationships.

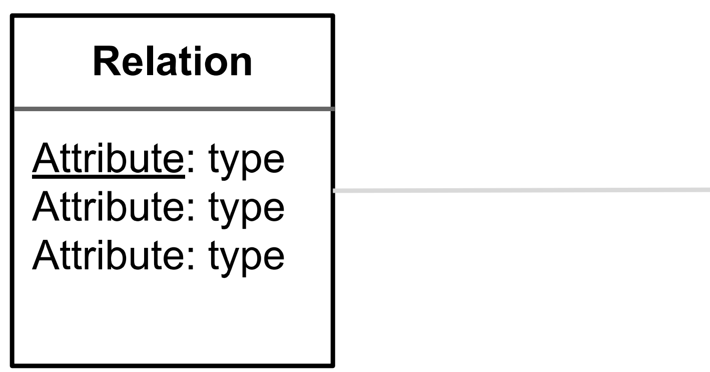
Communicating a Relational Model

- Rectangles represent tables (classes in your application).
- Attributes have data types: integer, decimal, boolean, string, date, time, timestamp, blob, list, enum.
  Underline the attribute(s) that uniquely identify an instance.
- Lines are relationships. A relationship is a link between classes or instances (binary association) [1].



Communicating a Relational Model

- Rectangles represent tables (classes in your application).
- Attributes have data types: integer, decimal, boolean, string, date, time, timestamp, blob, enum.
  <u>Underline</u> the attribute(s) that uniquely identify an instance.
- Lines are relationships. A relationship is a link between classes or instances (binary association) [1].



Communicating a Relational Model

- Rectangles represent tables (classes in your application).
- Attributes have data types: integer, decimal, boolean, string, date, time, timestamp, blob, list, enum.
  Underline the attribute(s) that uniquely identify an instance.
- Lines are relationships. A relationship is a link between classes or instances (binary association) [1].

1. Relationships in UMLs allow more than two endpoints, such as the ternary association, which is a relationship between three classes. For this course, we will only allow two endpoints. Anything more indicates your classes are not well encapsulated and/or too tightly coupled.

Cardinality

- The number of **instances** that participate in the relationship.
- Cardinality annotations:
  - 1: exactly one.
  - 0..1: zero or one [1].
  - *: zero or more (shorthand for 0..*).
  - 1..*: one or more.

1. The “0..1” cardinality is referred to “optionally”. So “0..1” is read as “optionally one”.

Relationship Cardinality

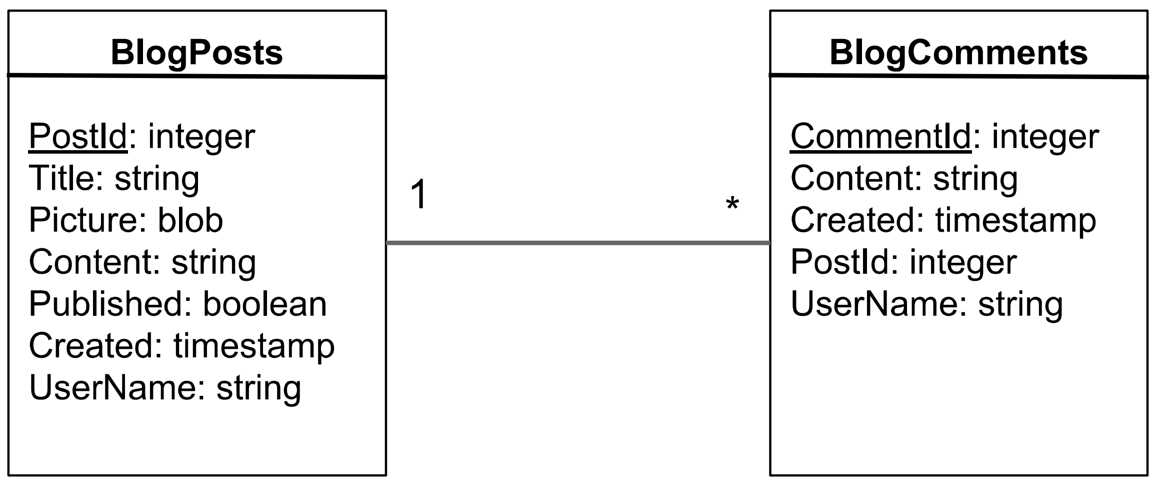
- **one-one**: 1 - 1. One instance of Class A is related to one instance of Class B. [1]
- **one-many**: 1 - *. One instance of Class A is related to multiple instances of Class B, *and* each instance of Class B is related to one instance of Class A.
- **many-many**: * - *. One instance of Class A is related to multiple instance of Class B, *and* one instance of Class B is related to multiple instance of Class A. [2]

1. 1 -1 is not common in a UML. When a 1 -1 relationship exists, the two classes can likely be combined.
2. We also will not see * - * in a normalized UML. See the slides about Reification.

Example: Cardinality

one-many: BlogPosts-BlogComments

A blog post can have any number of comments (zero or more).

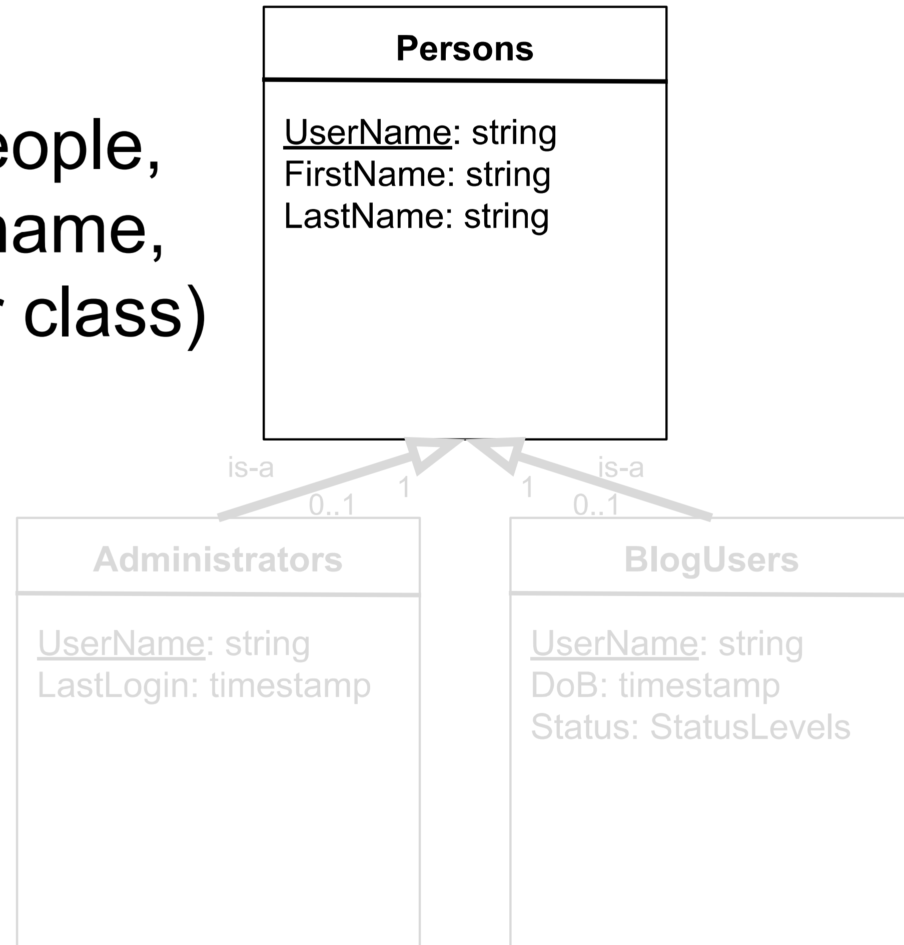


Generalization Relationship

- Relationship between **classes**.
- “is-a” relationship. Subclass inheritance of a super type.
- Triangle to annotate relationship. △

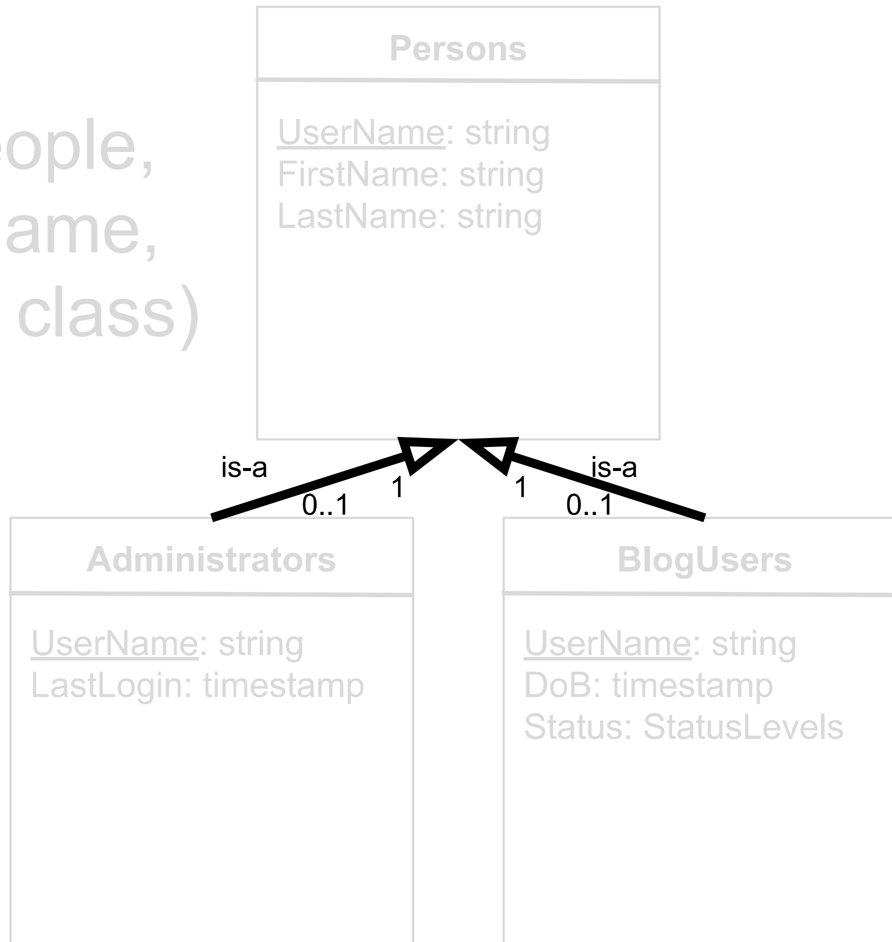
Example: Generalization

- Blog application consists of people, which have a first name, last name, and unique user name. (super class)
- Two specific types of people (sub classes):
  - Administrator, which as a last login time.
  - Blog user, which has a date of birth and status level.



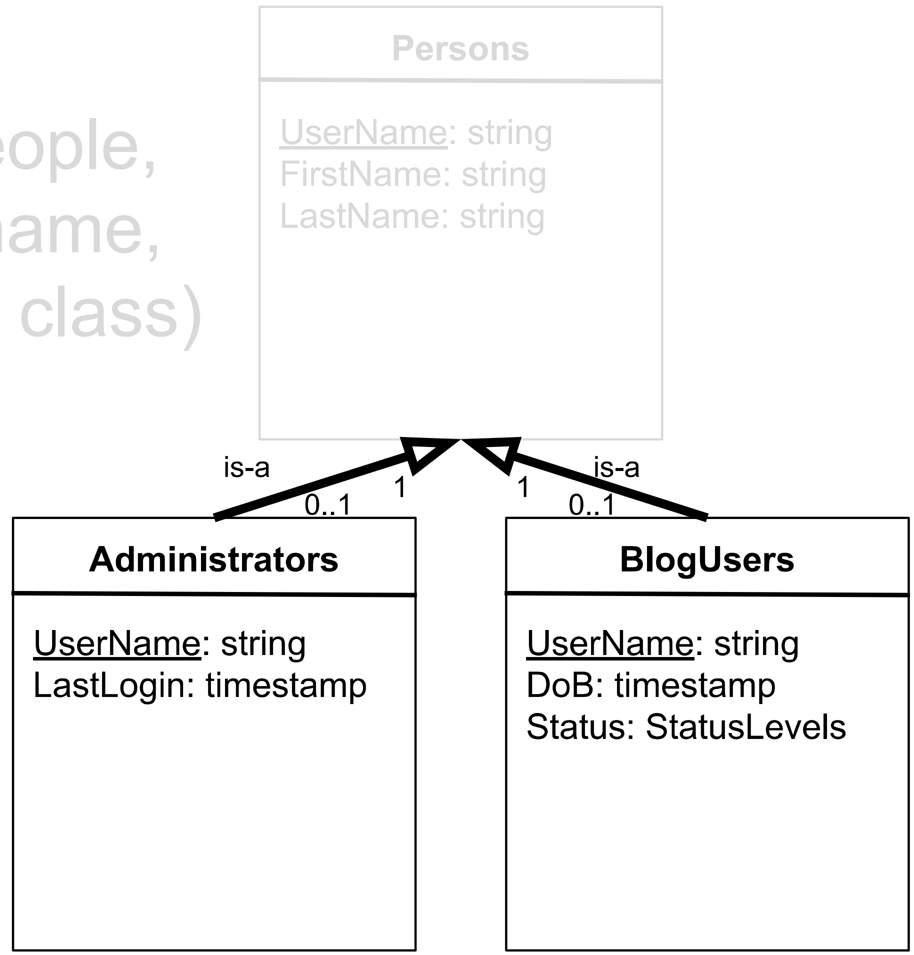
Example: Generalization

- Blog application consists of people, which have a first name, last name, and unique user name. (super class)
- Two specific types of people (sub classes):
  - Administrator, which as a last login time.
  - Blog user, which has a date of birth and status level.



Example: Generalization

- Blog application consists of people, which have a first name, last name, and unique user name. (super class)
- Two specific types of people (sub classes):
  - Administrator, which as a last login time.
  - Blog user, which has a date of birth and status level.

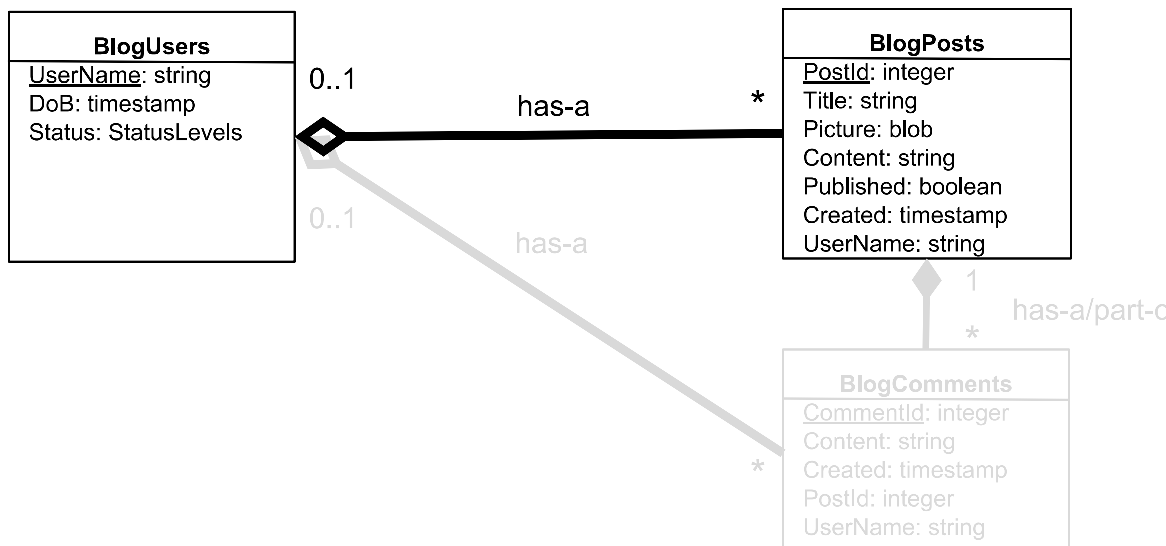


Aggregation/Composition Relationships

- Relationship between **instances**.
- **Aggregation**: “has-a”. No life cycle dependency. Can be a collection or container for other instances. Hollow diamond.◇
- **Composition**: “has-a”, but that is “part-of” a whole. Strong life cycle dependency. Solid diamond. ◆
- Difference is **life cycle dependency**. Can one exist without the other? If yes, then aggregation. If no (they either both must exist or both must not), then composition.

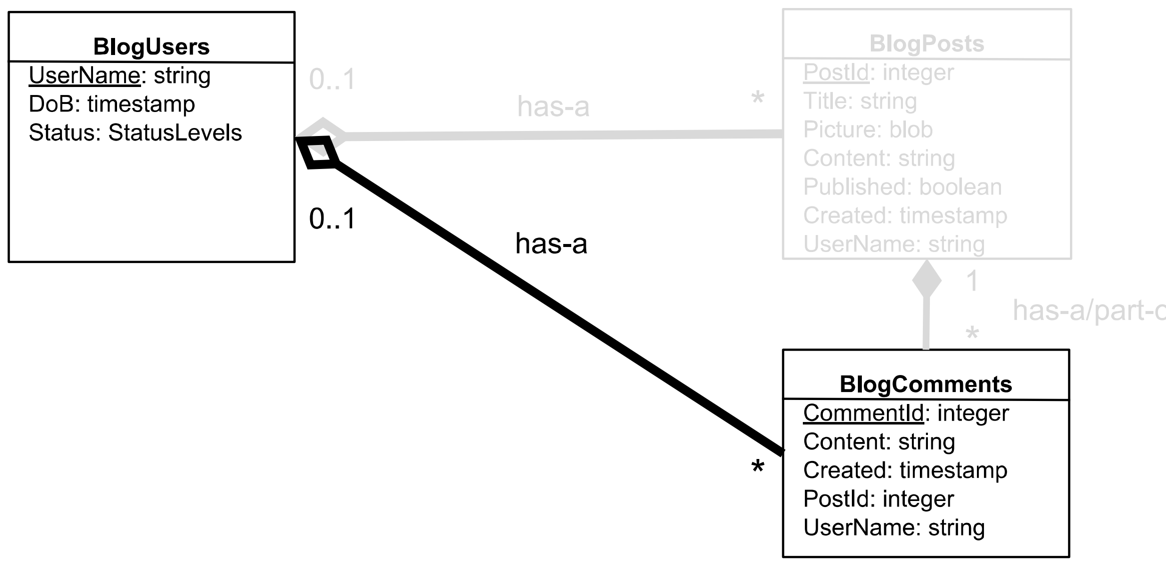
Example: Aggregation/Composition

- A blog user can publish any number of blog posts. When a user is deleted, their posts are not deleted.
- A blog user can create any number of blog comments. When a user is deleted, their comments are not deleted.
- A blog post can have any number of comments. When a post is deleted, then its comments are deleted.
- Recall the difference is **life cycle dependency**.



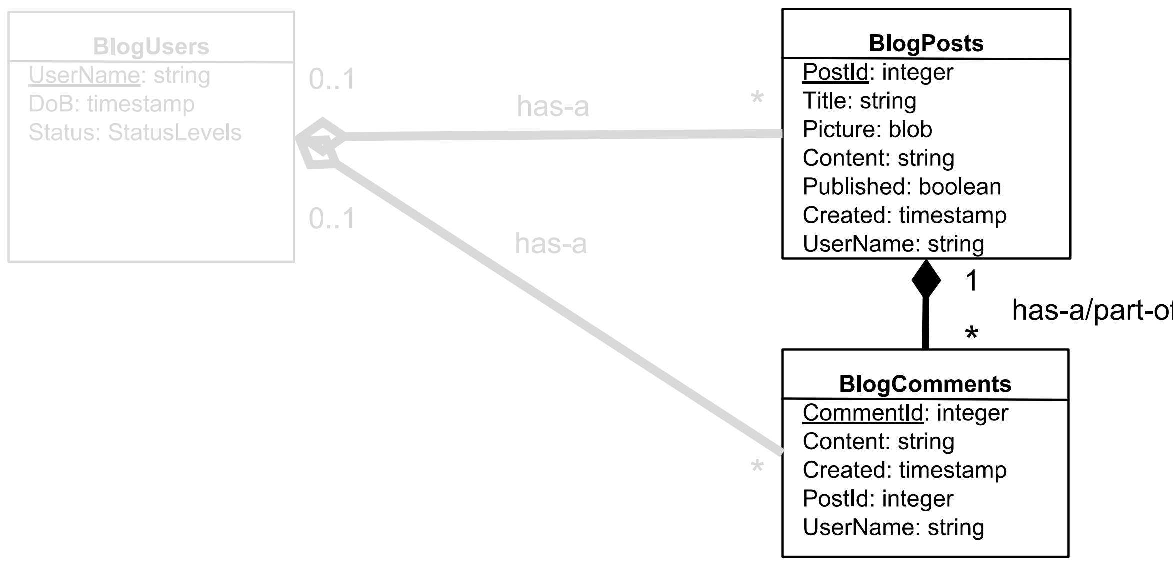
Example: Aggregation/Composition

- A blog user can publish any number of blog posts. When a user is deleted, their posts are not deleted.
- A blog user can create any number of blog comments. When a user is deleted, their comments are not deleted.
- A blog post can have any number of comments. When a post is deleted, then its comments are deleted.
- Recall the difference is **life cycle dependency**.



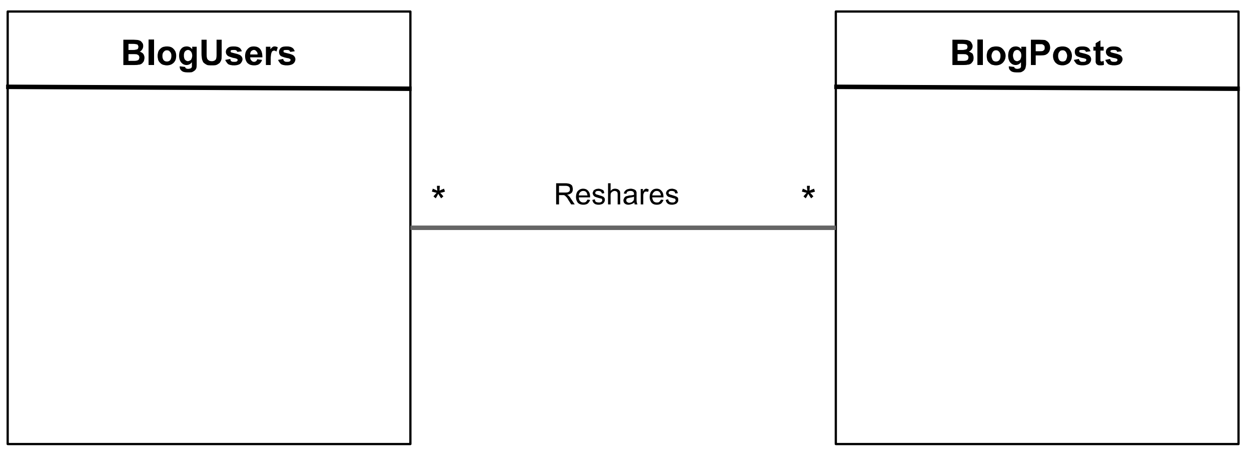
Example: Aggregation/Composition

- A blog user can publish any number of blog posts. When a user is deleted, their posts are not deleted.
- A blog user can create any number of blog comments. When a user is deleted, their comments are not deleted.
- A blog post can have any number of comments. When a post is deleted, then its comments are deleted.
- Recall the difference is **life cycle dependency**.



Association Relationship

- Relationship between **instances**.
- Many-many.
- Example: any number of blog users can reshare any number of blog posts.



Example: Association

- Problem: what do records in a database look like for many-many relationships?
- Should BlogUsers have a list of BlogPosts, and should BlogPosts have a list of BlogUsers instances?

BlogUsers	
UserName	Reshares
Jae	1,2,3
Sue	2,3

BlogPosts	
PostId	Reshares
1	Jae
2	Jae,Sue
3	Jae,Sue

Example: Association

- Jae unshares PostId 3.
- Requires update in two tables.
- If only one table is updated, then what is the source of truth?

BlogUsers	
UserName	Reshares
Jae	1,2,~~3~~
Sue	2,3

BlogPosts	
PostId	Reshares
1	Jae
2	Jae,Sue
3	~~Jae~~,Sue

Example: Association

- What if reshares is only in the BlogUsers table.
- How do we add more information to a reshare, such as the timestamp of each reshare?

BlogUsers	
UserName	Reshares
Jae	1,2,3
Sue	2,3

BlogPosts
PostId
1
2
3

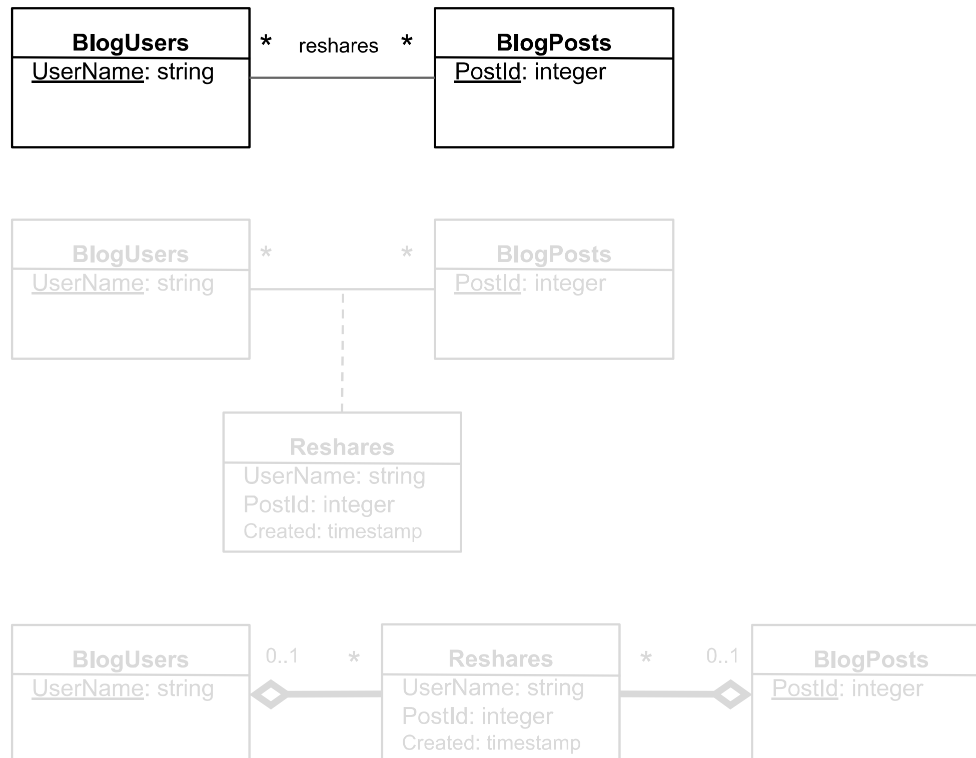
Reification

- Use an **association class** to represent the many-many relationship. This process is known as “reifying the association relationship”or “reification” [1].

1. Definition of reify is to “make something concrete”.

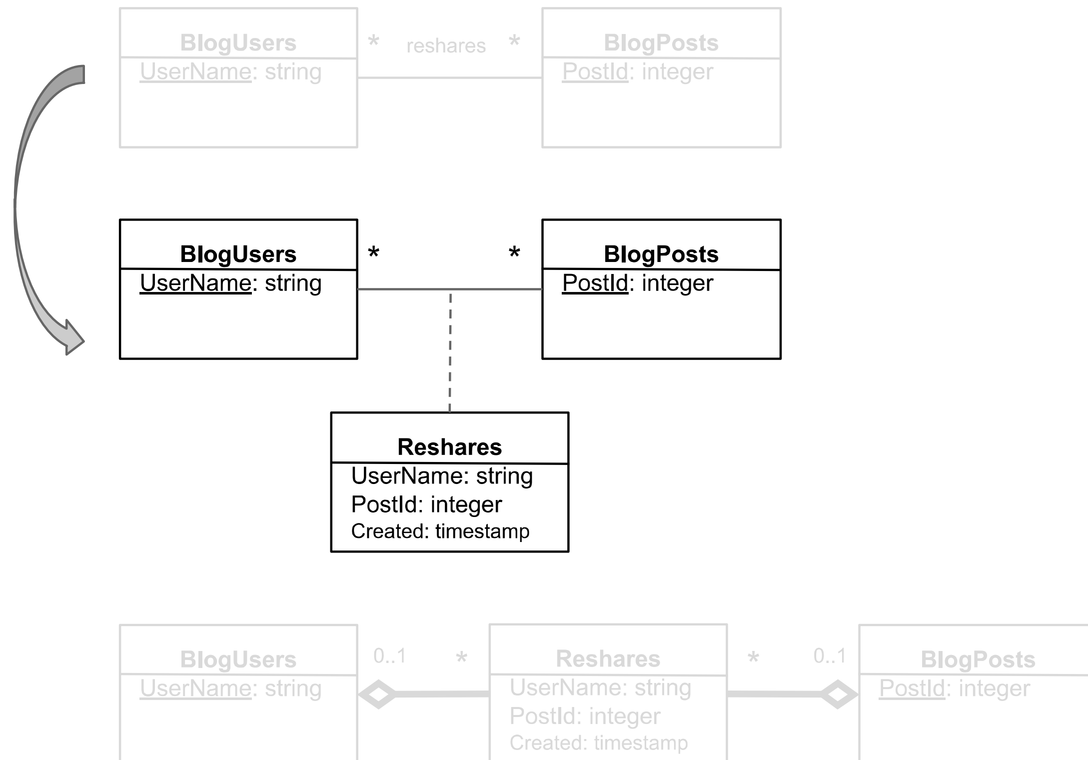
Example: Reification

1. Describe the many-many relationship.
2. Introduce association class (dotted line), and add attributes that further describe the relationship.
3. Convert association relationship to one-many aggregation or composition.



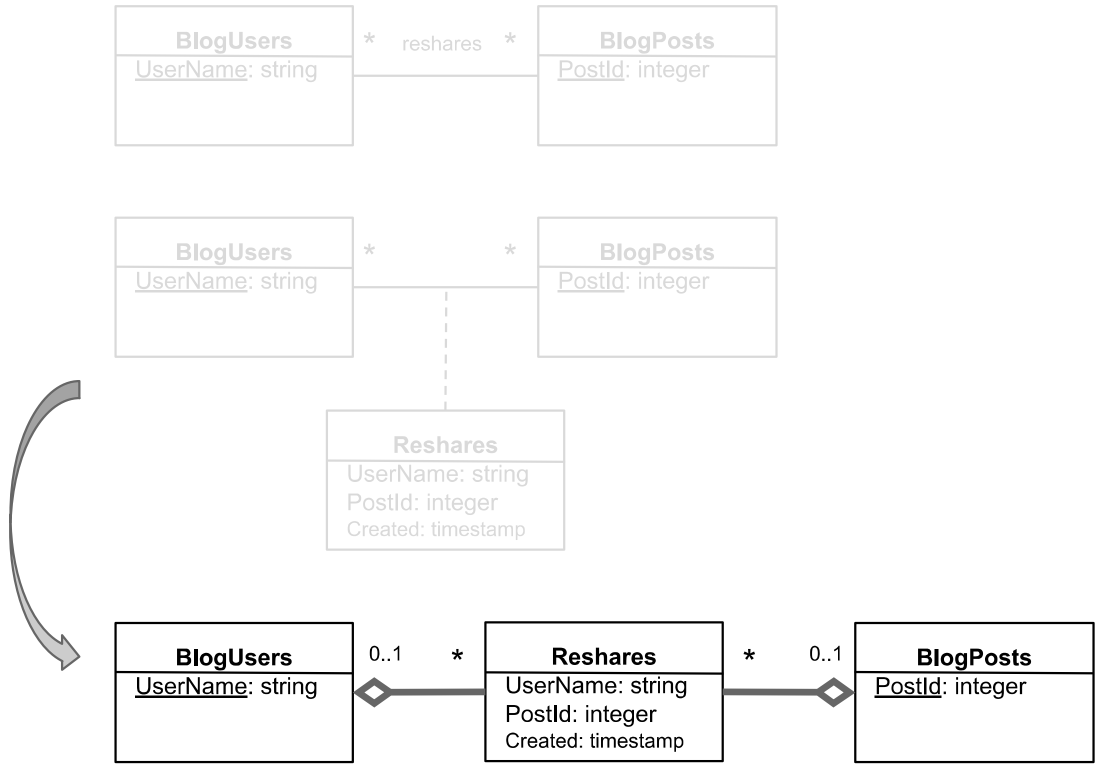
Example: Reification

1. Describe the many-many relationship.
2. Introduce association class (dotted line), and add attributes that further describe the relationship.
3. Convert association relationship to one-many aggregation or composition.



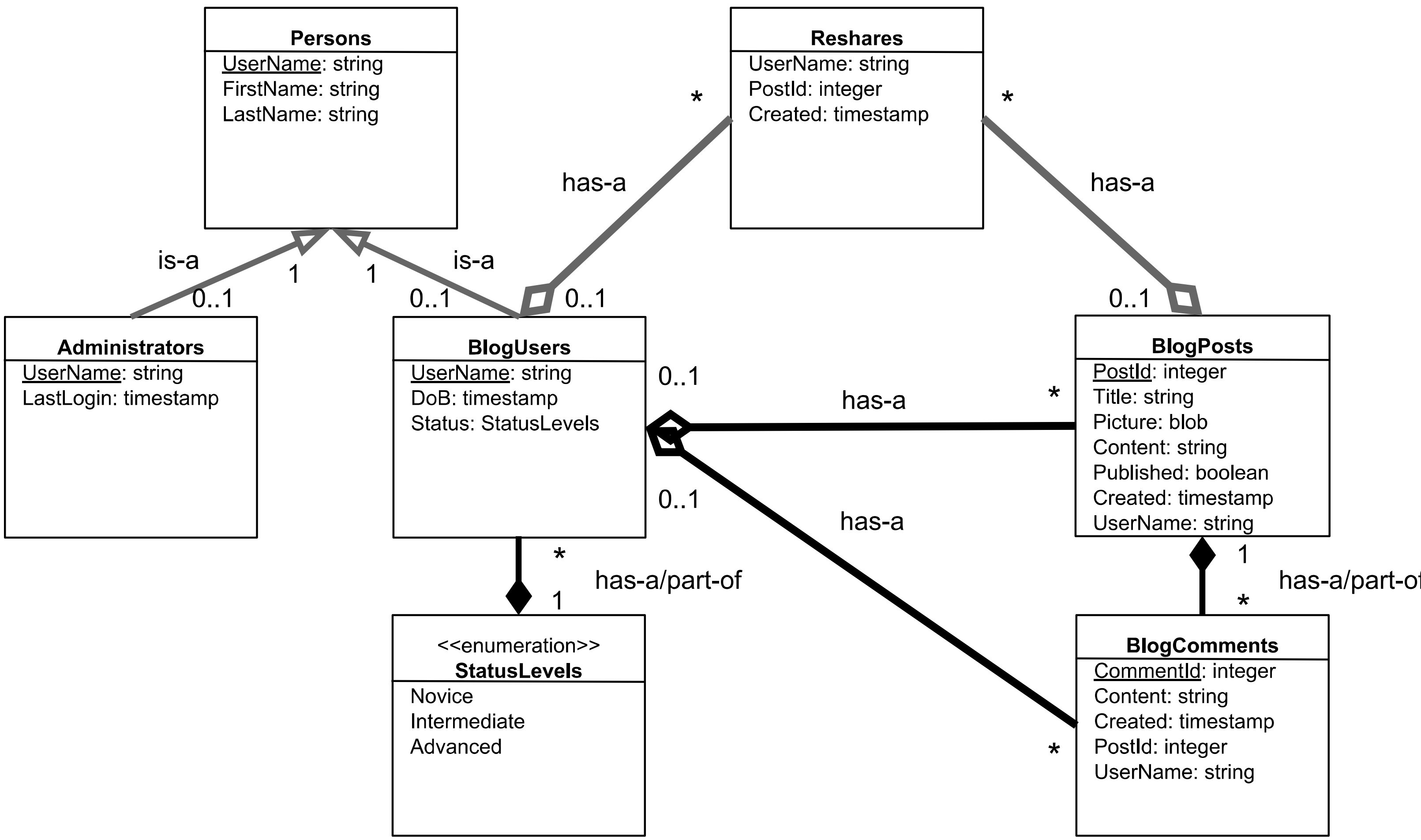
Example: Reification

1. Describe the many-many relationship.
2. Introduce association class (dotted line), and add attributes that further describe the relationship.
3. Convert association relationship to one-many aggregation or composition.



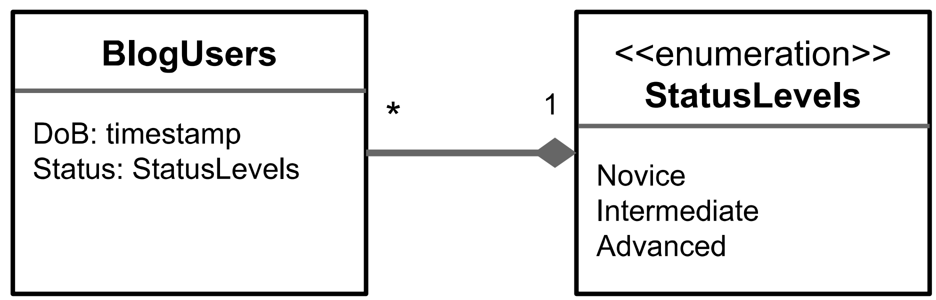
Should we use an aggregation or composition? Recall life **cycle dependency**. This example states that reshare records will continue to exist when blog user or blog post records are deleted.

Example: Blog Application UML



Enumeration Note

- Although in some UMLs enum relationship can be a general dependency using an arrow (→), we try to be more specific with an aggregation/composition:



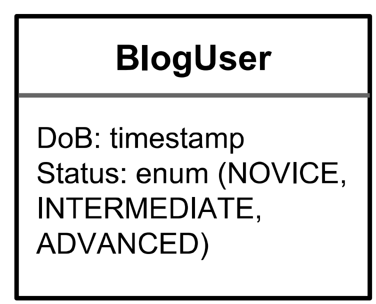
Note that StatusLevels lists records, not attributes.

- Note we will also see the following shorthand as we become familiar with the MySQL enum datatype:

Enumeration Note

- Although in some UMLs enum relationship can be a general dependency using an arrow (→), we try to be more specific with an aggregation/composition:

- Note we will also see the following shorthand as we become familiar with the MySQL enum datatype:



Relational Databases

L3: Alternative Data Modeling

Alternative: Entity-Relationship Diagram

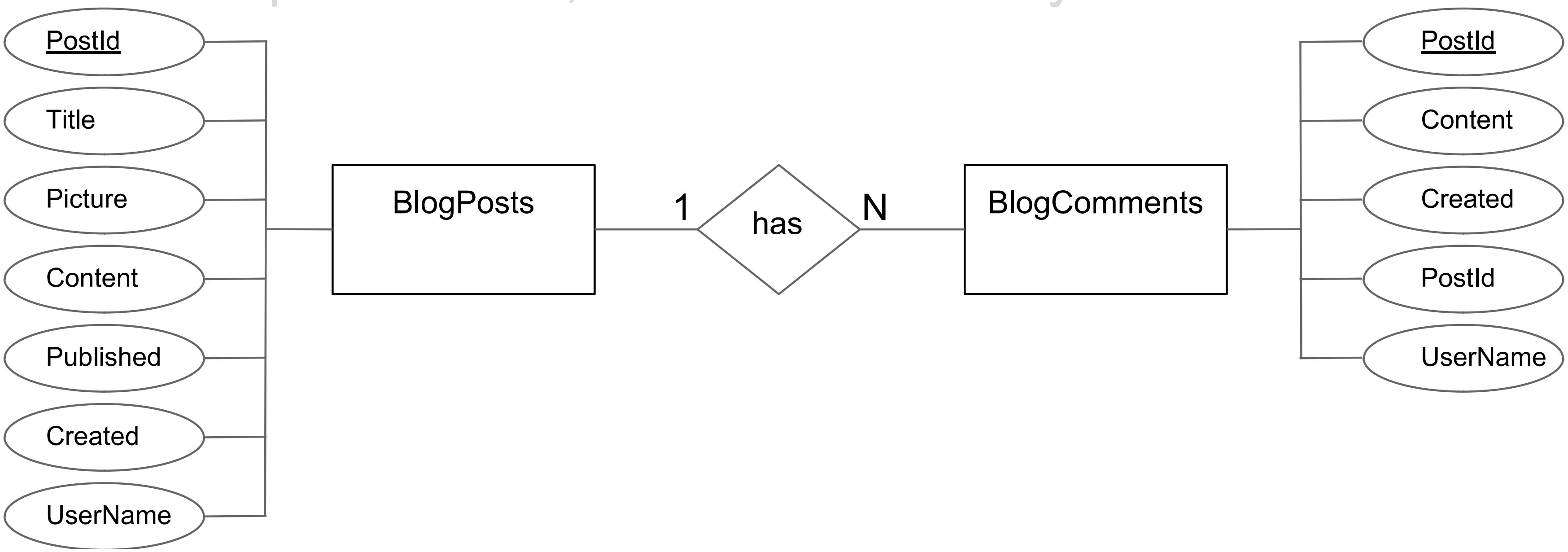
- Entity-Relationship (ER) diagrams are also commonly used for data modeling.
- Why we choose UML over ER:
  - Relevance to our class: structural UML can be modified to communicate dynamic behavior in OO design.
  - UML can easily represent data lifecycle.
  - ER complexity: multiple “levels”.
  - ER complexity: multiple “notations”.

ER Diagram Level

- Conceptual: entities, relationships.
- Logical: entities, relationships, attributes.
- Physical: entities, relationships, attributes, data types.

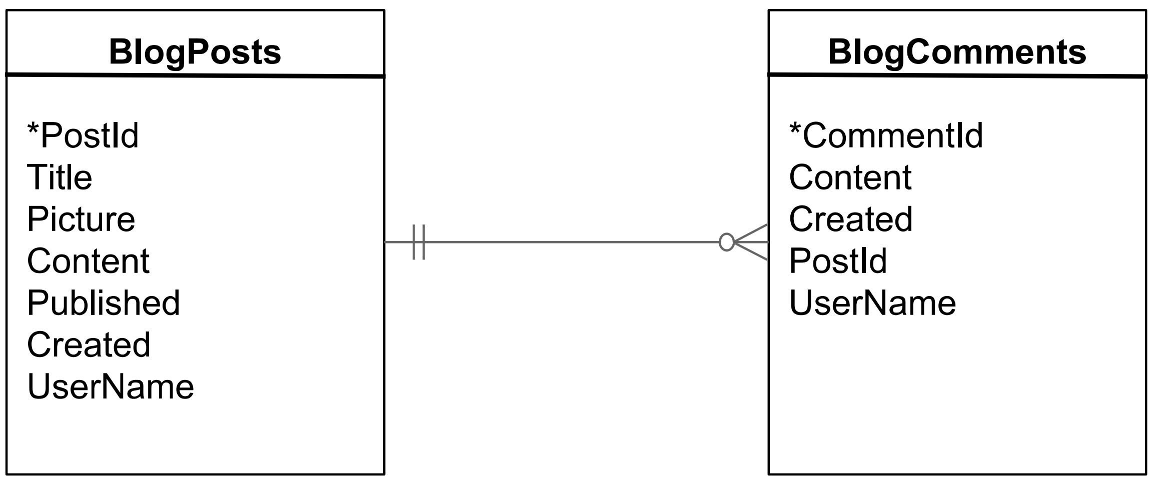
ER Diagram Notation

- Chen: entities are rectangles, attributes are ovals, relationships are lines with diamonds, cardinalities are numbers.
- Crow's foot: entities are rectangles, attributes are text in rectangles, relationships are lines, cardinalities are symbols on lines.



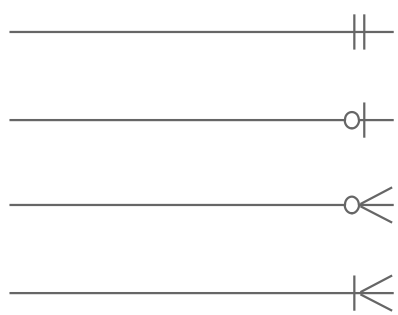
ER Diagram Notation

- Chen: entities are rectangles, attributes are ovals, relationships are lines with diamonds, cardinalities are numbers.
- Crow's foot: entities are rectangles, attributes are text in rectangles, relationships are lines, cardinalities are symbols on lines.



ER Diagram Notation

- Chen: entities are rectangles, attributes are ovals, relationships are lines with diamonds, cardinalities are numbers.
- Crow’s foot: entities are rectangles, attributes are text in rectangles, relationships are lines, cardinalities are symbols on lines.



Crow’s foot		UML
	Exactly 1	1
	Zero or 1	0..1
	Zero or more	*
	1 or more	1..*