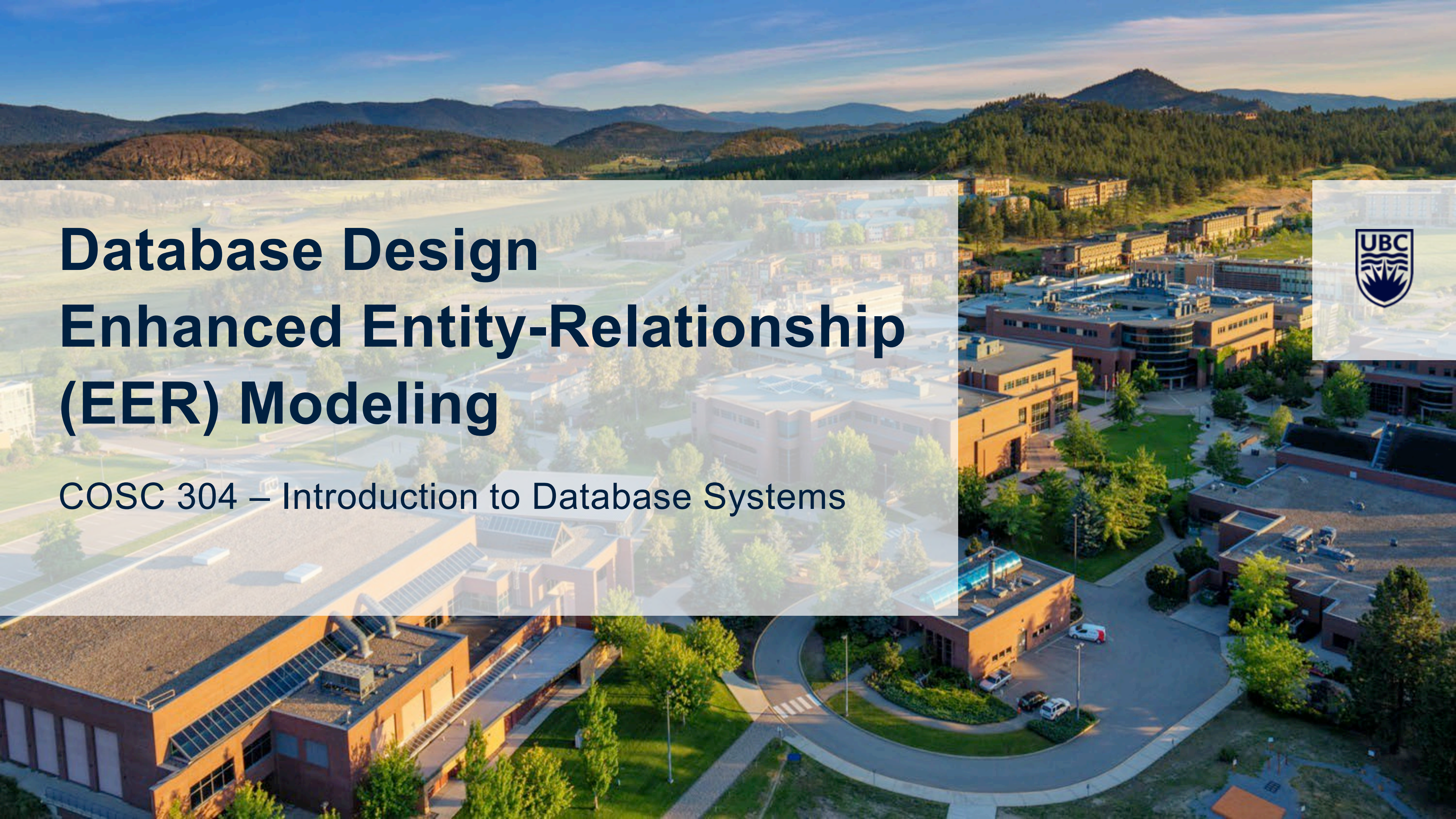


Database Design Enhanced Entity-Relationship (EER) Modeling

COSC 304 – Introduction to Database Systems

Enhanced Entity-Relationship Modeling

Enhanced Entity-Relationship (EER) modeling is an extension of ER modeling to include object-oriented concepts such as:

- superclasses and subclasses
- specialization and generalization
- aggregation and composition

These modeling constructs may allow more precise modeling of systems that are object-oriented in nature such as:

- GIS (Geographical Information Systems)
- CAD/CAM systems (Computer-Aided Design/Manufacturing)

Review: Superclasses and Subclasses

The object-oriented ideas of inheritance and superclasses and subclasses are taught during programming in an OO language such as Java.

A ***superclass*** is a general class that is extended by one or more subclasses.

A ***subclass*** is a more specific class that extends a superclass by inheriting its methods and attributes and then adding its own methods and attributes.

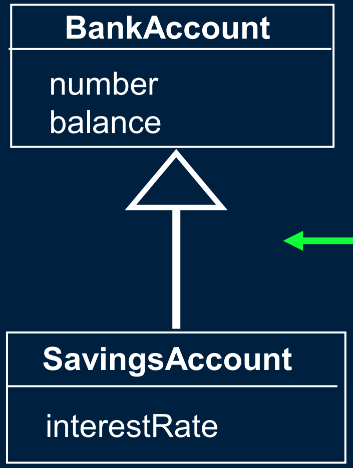
Inheritance is the process of a subclass inheriting all the methods and attributes of a superclass.

Superclasses and Subclasses Example

Java code:

```
public class SavingsAccount extends BankAccount
```

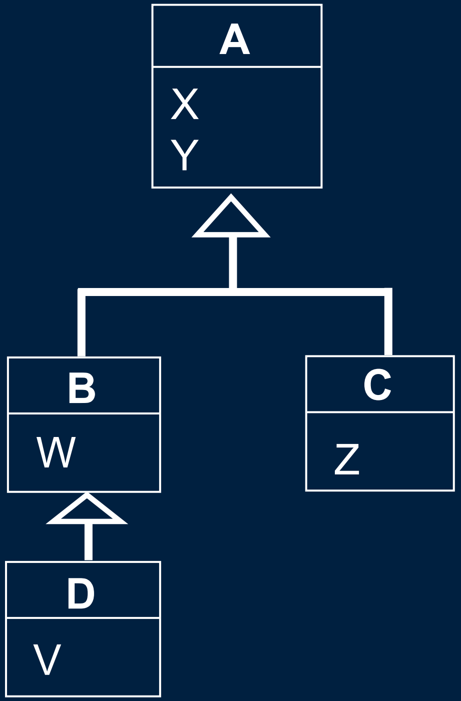
UML class diagram:



Triangle points to superclass

Superclasses and Subclasses Question

Question: How many of the following statements are ***true***?



1) D is a superclass.

2) D has 1 attribute (including inherited attributes).

3) B and C are subclasses of A.

4) B inherits V from D.

5) D inherits attribute X.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

When to use EER Modeling?

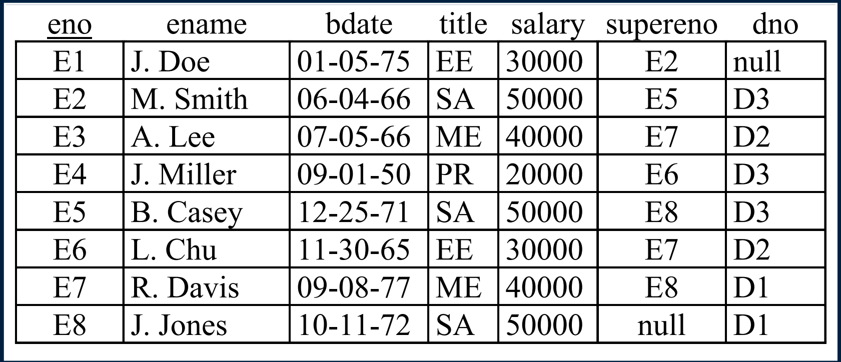
Many database projects do not need the object-oriented modeling features of EER modeling.

EER modeling is useful when the domain being modeled is object-oriented in nature and inheritance reduces the complexity of the design. Common cases:

- 1) When using *attribute inheritance* can reduce the use of nulls in a single entity relation (that contains multiple subclasses).
- 2) Subclasses can be used to explicitly model and name subsets of entity types that participate in their own relationships.

When to use EER Modeling? Using Attribute Inheritance

Emp Relation



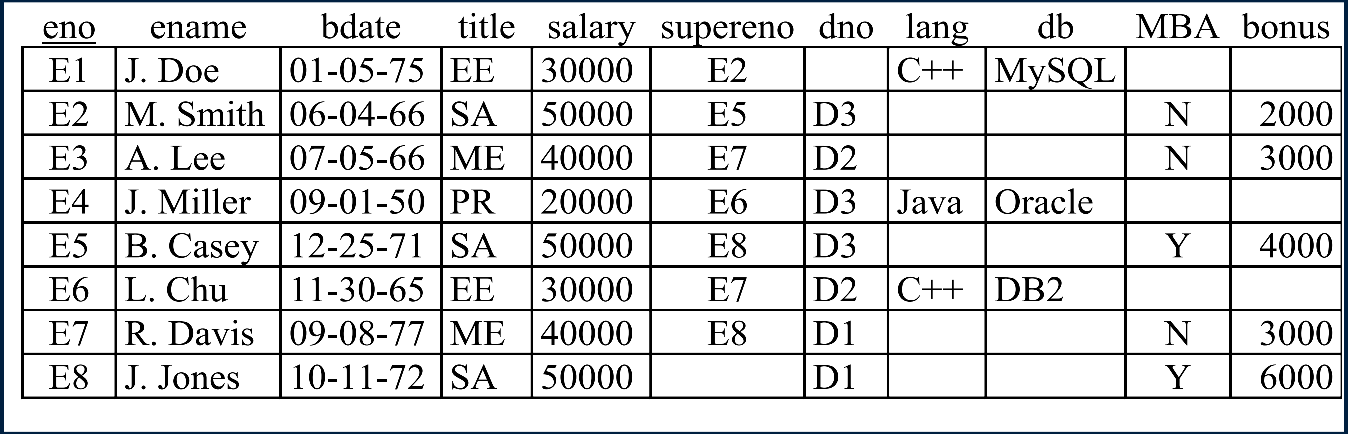
eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Note that the `title` attribute indicates what job the employee does at the company. Consider if each job title had its own unique information that we would want to record such as:

- `EE`, `PR` - programming language used (`lang`), DB used (`db`)
- `SA`, `ME` - MBA? (`MBA`), `bonus`

When to use EER Modeling? Using Attribute Inheritance (2)

We could represent all these attributes in a single relation:

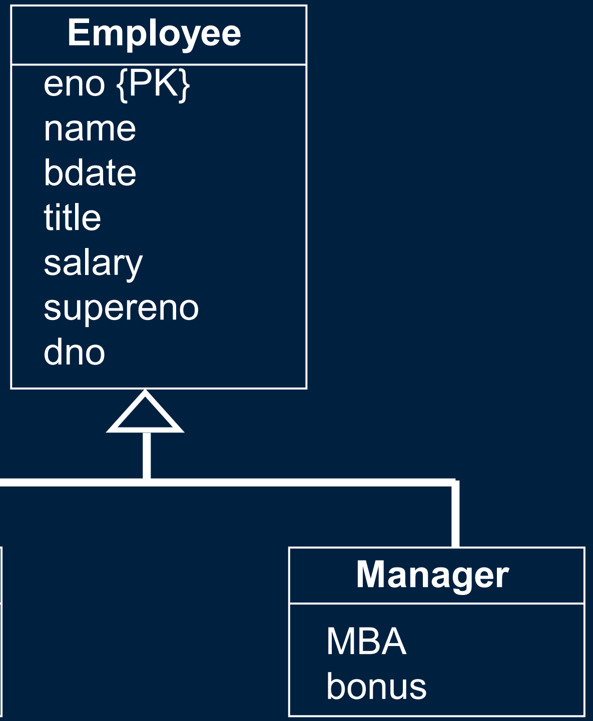


eno	ename	bdate	title	salary	supereno	dno	lang	db	MBA	bonus
E1	J. Doe	01-05-75	EE	30000	E2		C++	MySQL		
E2	M. Smith	06-04-66	SA	50000	E5	D3			N	2000
E3	A. Lee	07-05-66	ME	40000	E7	D2			N	3000
E4	J. Miller	09-01-50	PR	20000	E6	D3	Java	Oracle		
E5	B. Casey	12-25-71	SA	50000	E8	D3			Y	4000
E6	L. Chu	11-30-65	EE	30000	E7	D2	C++	DB2		
E7	R. Davis	09-08-77	ME	40000	E8	D1			N	3000
E8	J. Jones	10-11-72	SA	50000		D1			Y	6000

Note the wasted space as attributes that do not apply to a particular subclass are `NULL`.

When to use EER Modeling? Using Attribute Inheritance (3)

A better solution would be to make two subclasses of `Employee` called `Developer` and `Manager`:



When to use EER Modeling? Using Attribute Inheritance (4)

Resulting relations:

Employee Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Developer Relation

eno	lang	db
E1	C++	MySQL
E4	Java	Oracle
E6	C++	DB2

Manager Relation

eno	MBA	bonus
E2	N	2000
E3	N	3000
E5	Y	4000
E7	N	3000
E8	Y	6000

Generalization and Specialization

Subclasses and superclasses are created by using either generalization or specialization.

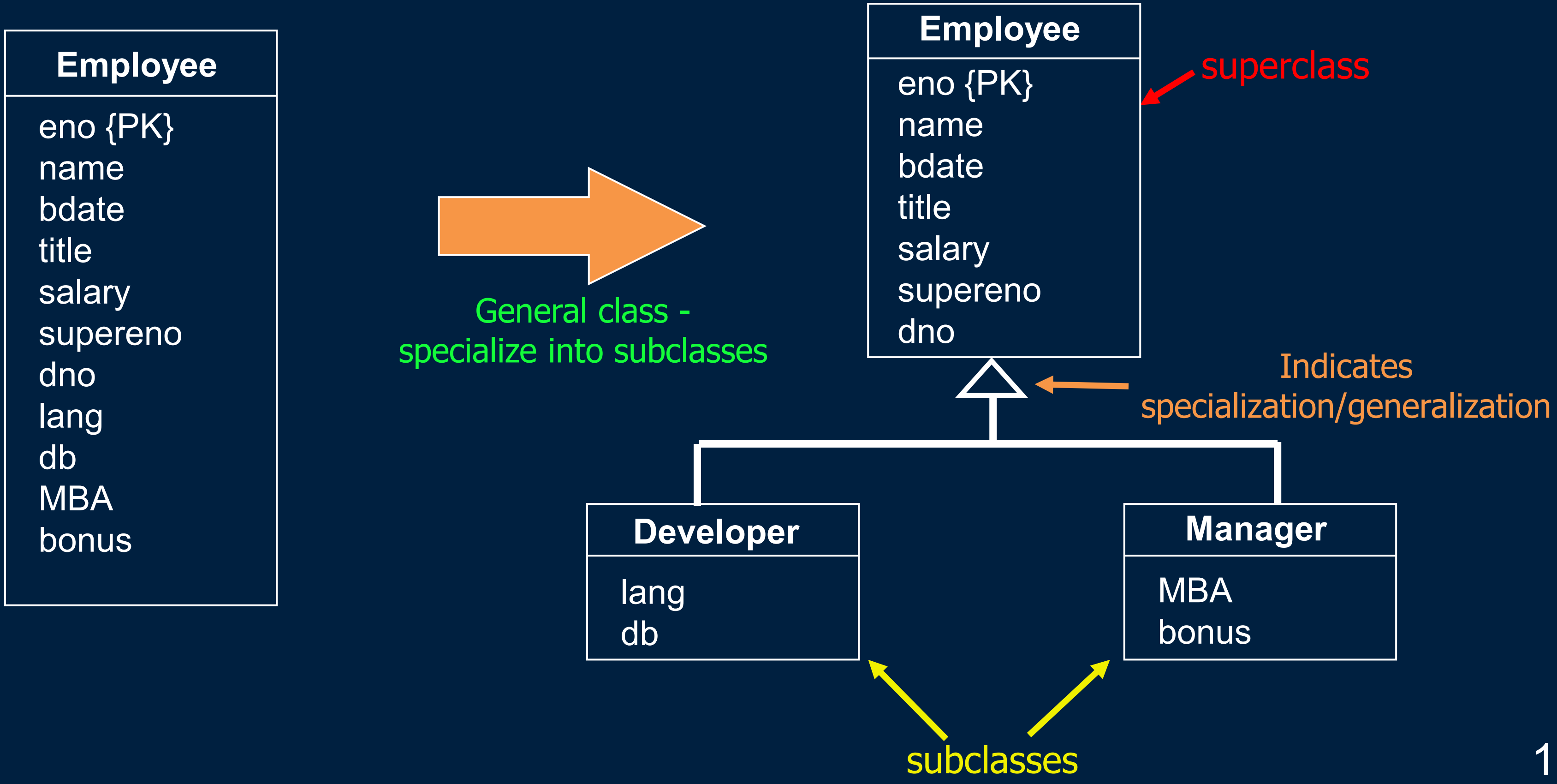
Specialization is the process of creating more specialized subclasses of an existing superclass.

- Top-down process: Start with a general class and then subdivide it into more specialized classes.
  - The specialized classes may contain their own attributes. Attributes common to all subclasses remain in the superclass.

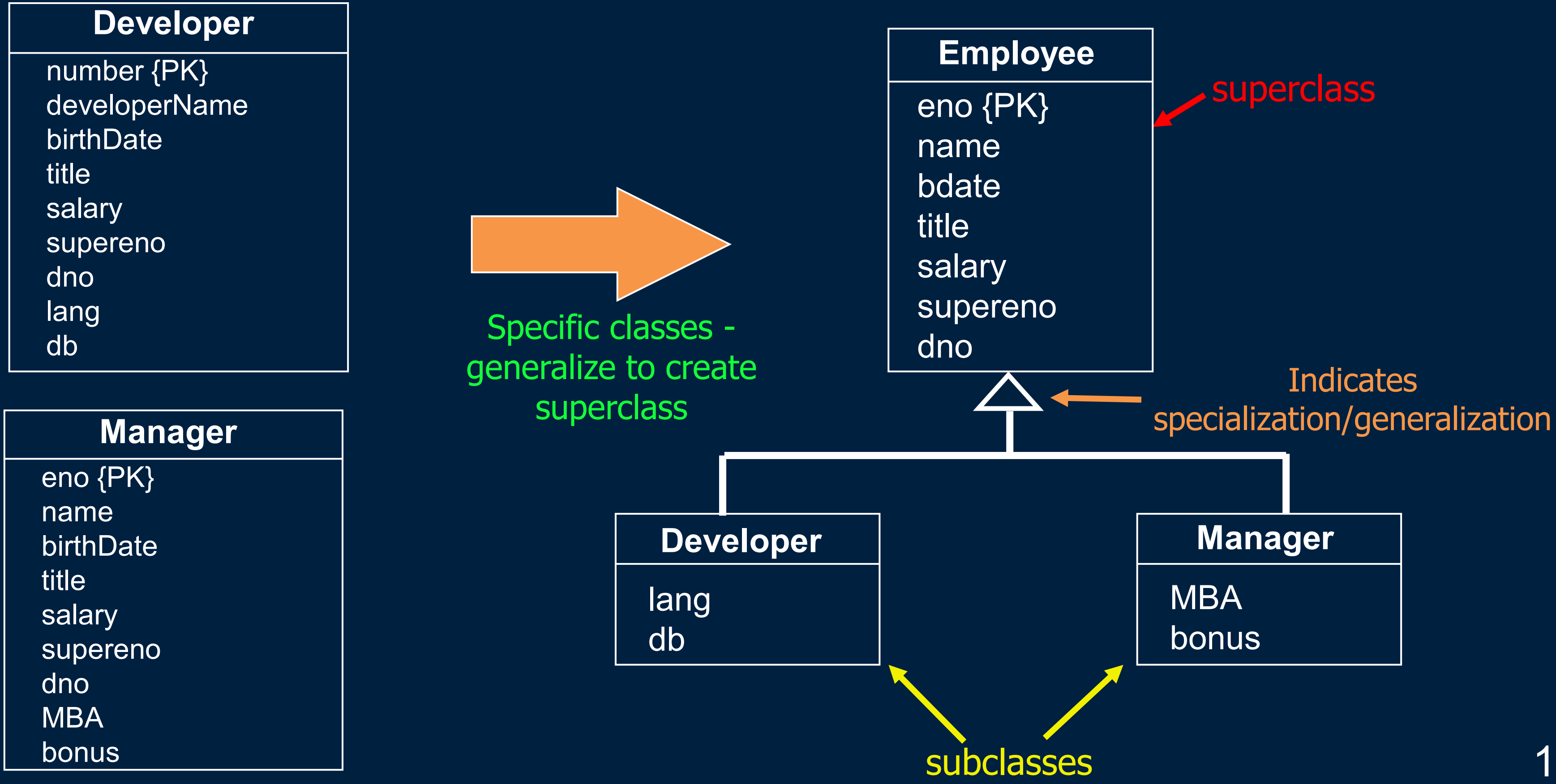
Generalization is the process of creating a more general superclass from existing subclasses.

- Bottom-up process: Start with specialized classes and try to determine a general class that contains the attributes common to all of them.

Specialization Example



Generalization Example



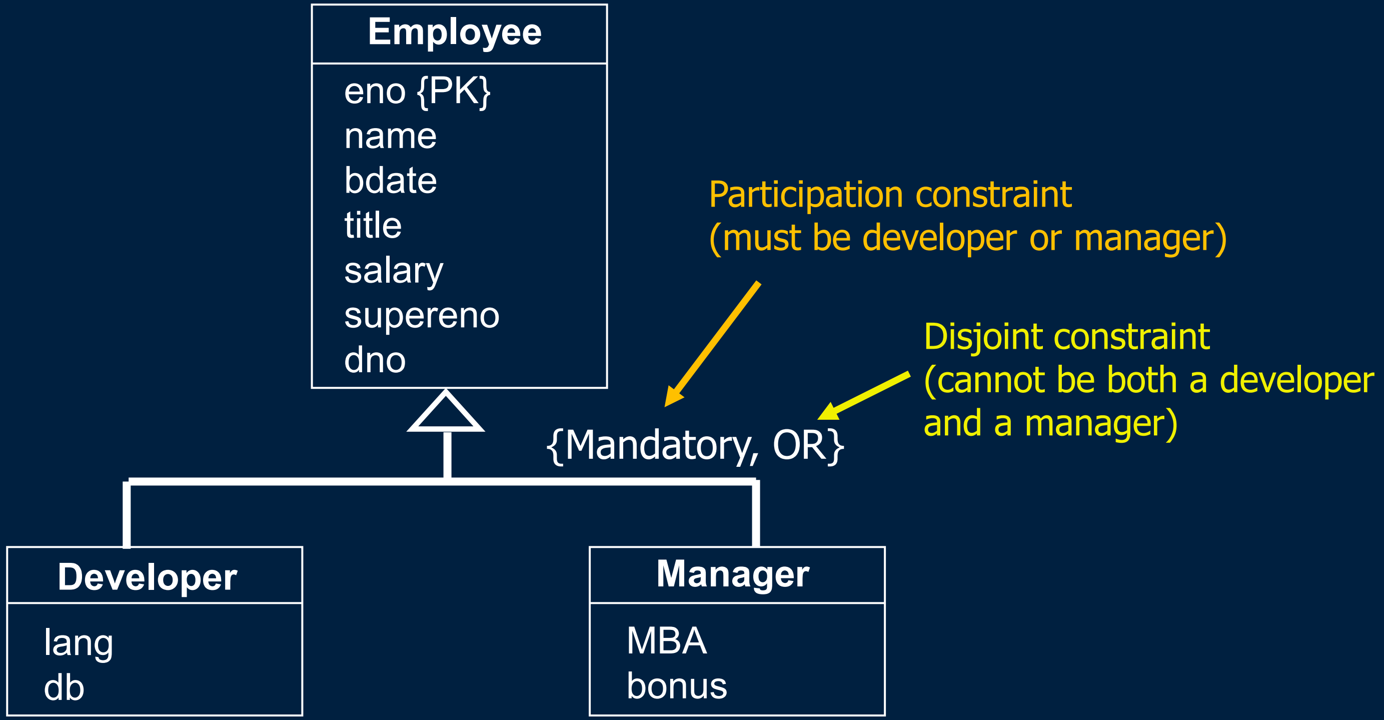
Constraints on Generalization and Specialization

There are two types of constraints associated with generalization and specialization:

- ***Participation constraint*** - determines if every member in a superclass must participate as a member of one of its subclasses.
  - It may be `optional` for a superclass member to be a member of one of its subclasses, or it may be `mandatory` that a superclass member be a member of one of its subclasses.
- ***Disjoint constraint*** - determines if a member of a superclass can be a member of one or more than one of its subclasses.
  - If a superclass object may be a member of only one of its subclasses this is denoted by `OR` (subclasses are *disjoint*).
  - Otherwise, `AND` is used to indicate that it may be in more than one of its subclasses.

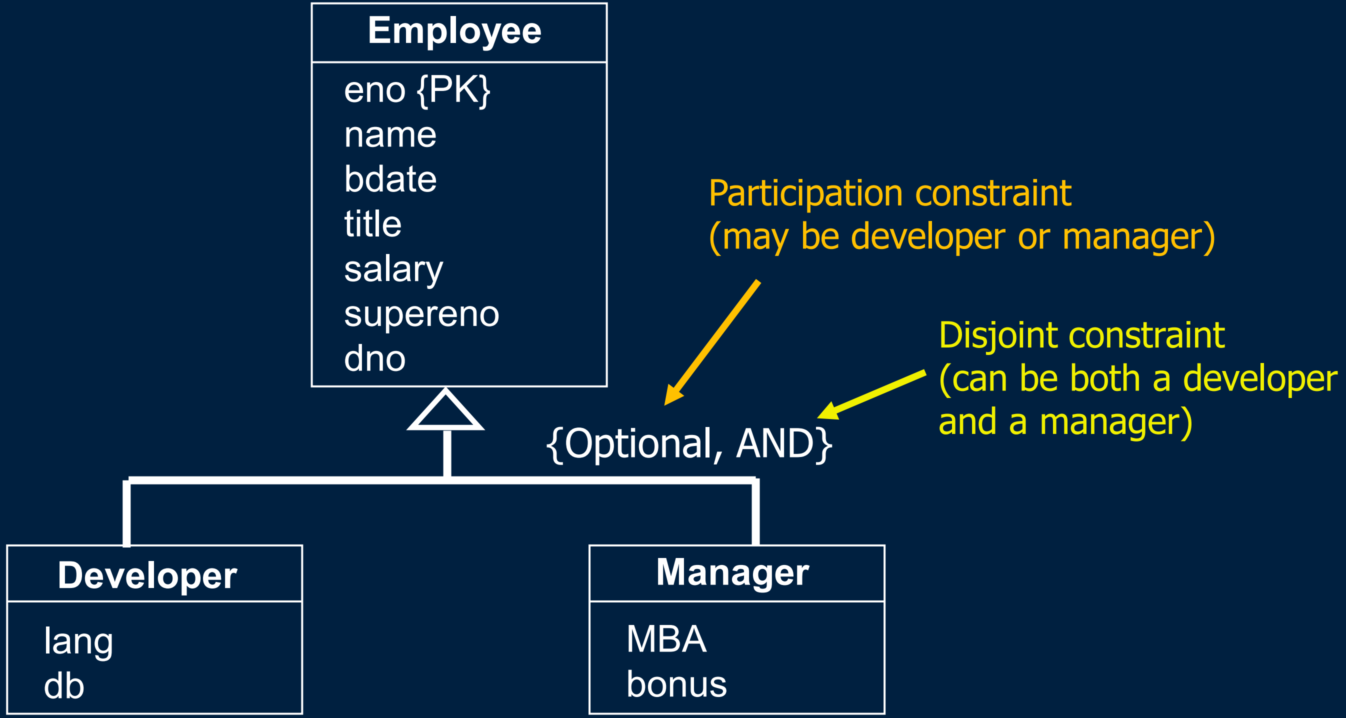
Constraints Example

An employee must be either a developer or a manager, but cannot be both.



Constraints Example (2)

An employee may specialize as a developer or manager. An employee may be both a manager and developer.



General Predicate Constraints

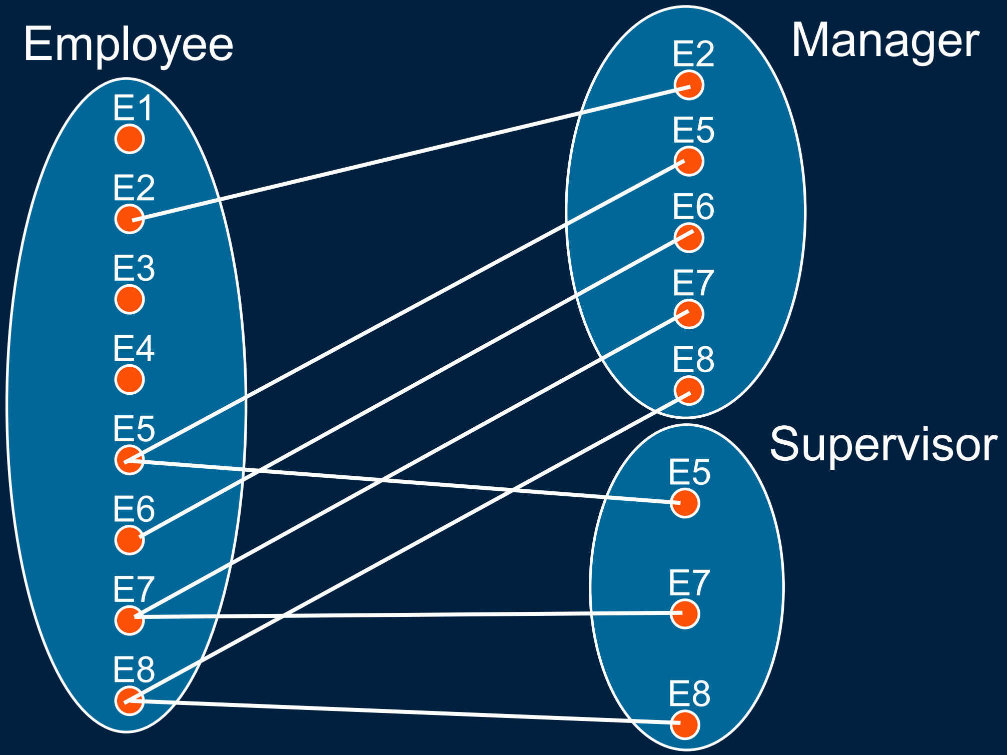
Predicate-defined constraints specify when an object participates in a subclass using a certain rule.

- For example, a subclass called `RichEmployees` can be defined with a membership predicate such as `salary >100000`.

Attribute-defined subclasses are a particular type of predicate-defined constraint where the value of an attribute(s) determines if an object is a member of a subclass.

- For example, the `title` field could be used as a *defining attribute* for the `Developer` and `Manager` subclasses.
  - `Emp` is in `Developer` if `title = 'EE' or 'PR'`
  - `Emp` is in `Manager` if `title = 'ME' or 'SA'`

Constraints Question



Note: What is the participation and the disjoint constraints for superclass `Employee` (with subclasses `Manager` and `Supervisor`) given these instances?

Relationship Constraints vs. Inheritance Constraints

There is a parallel between relationship constraints on relationships and inheritance constraints on superclasses and subclasses.

- Minimum # of occurrences – called participation constraint in both cases
- Maximum # of occurrences – called cardinality constraint for relationships and disjoint constraint for subclasses

Possible combinations:

Subclass Constraints	Relationship Constraints
Optional, AND	0..*
Optional, OR	0..1
Mandatory, AND	1..*
Mandatory, OR	1..1

EER Question

Question: How many of the following statements are ***true***?

1) Generalization is a bottom-up process.

2) In an UML diagram, the inheritance arrow points towards the superclass.

3) OPTIONAL and MANDATORY are possible choices for the participation constraint.

4) If the disjoint constraint is AND, a given object can be a member of multiple subclasses.

5) If the participation constraint is OPTIONAL, the disjoint constraint must be AND.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Multiple Inheritance

If each class only has one superclass, then the class diagram is said to be a ***specialization*** or ***type hierarchy***.

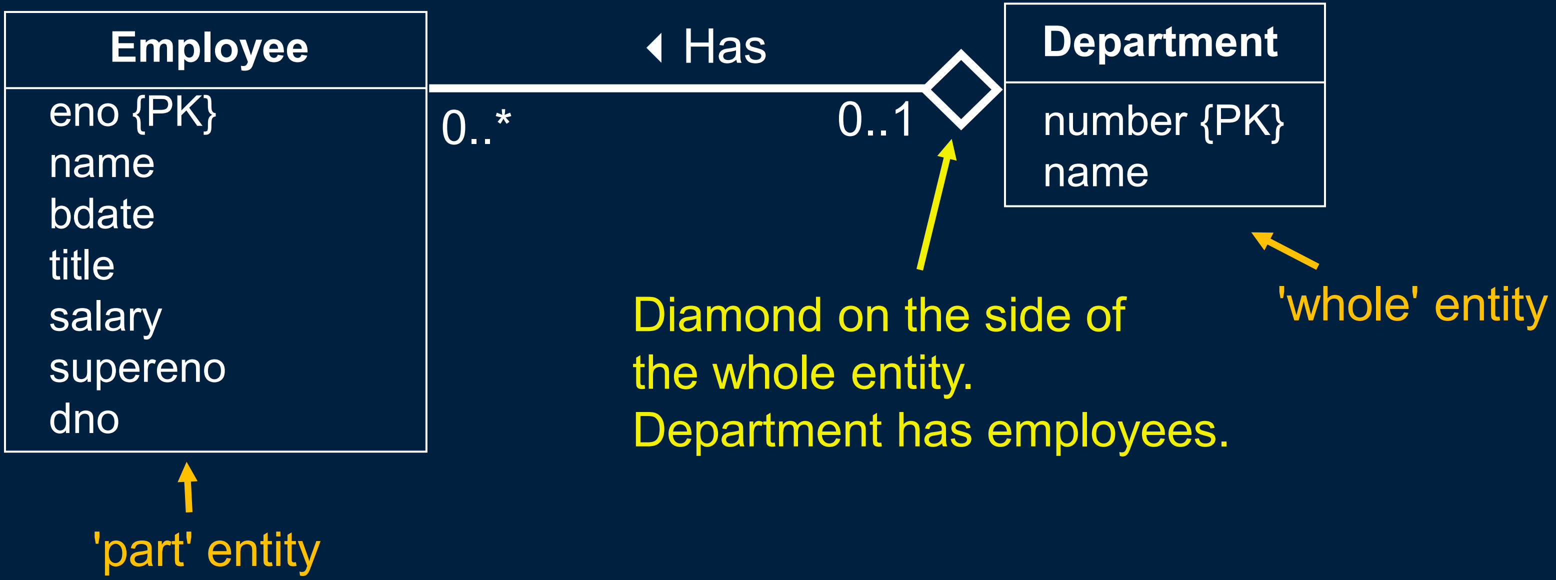
If a class may have more than one superclass, then the class diagram is said to be a ***specialization*** or ***type lattice***.

Although multiple inheritance is powerful, it should be avoided if possible.

Aggregation

Aggregation represents a `'HAS-A'` or `'IS-PART-OF'` relationship between entity types. One entity type is the *whole*, the other is the *part*.

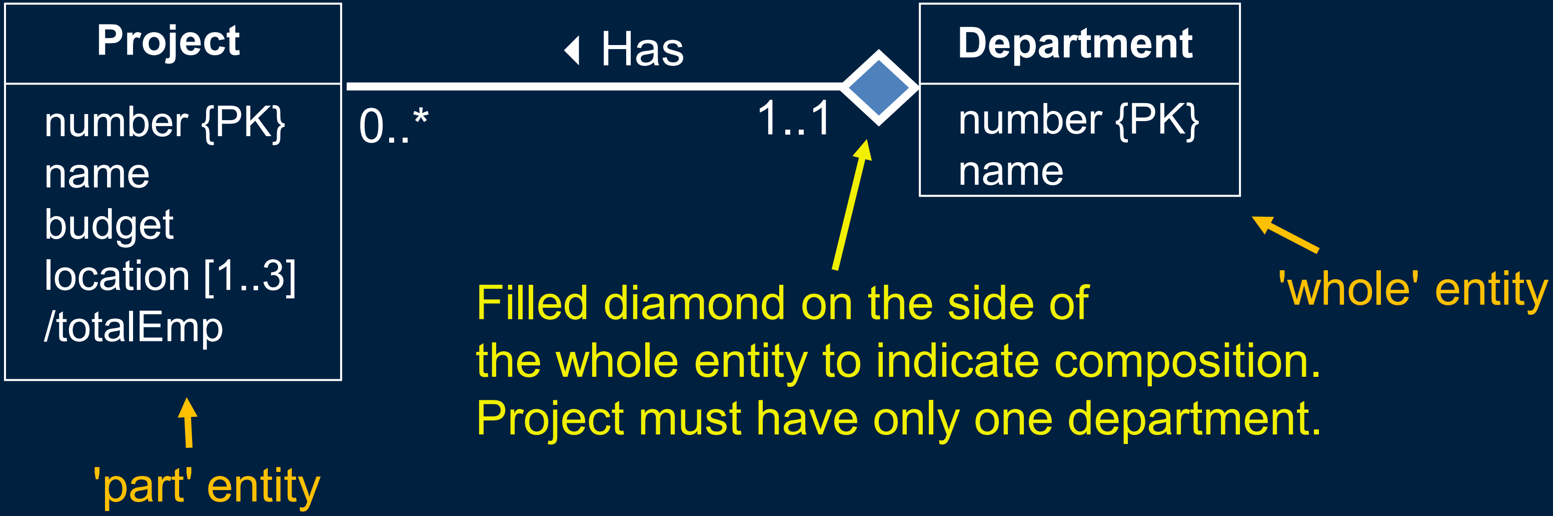
Example:



Composition

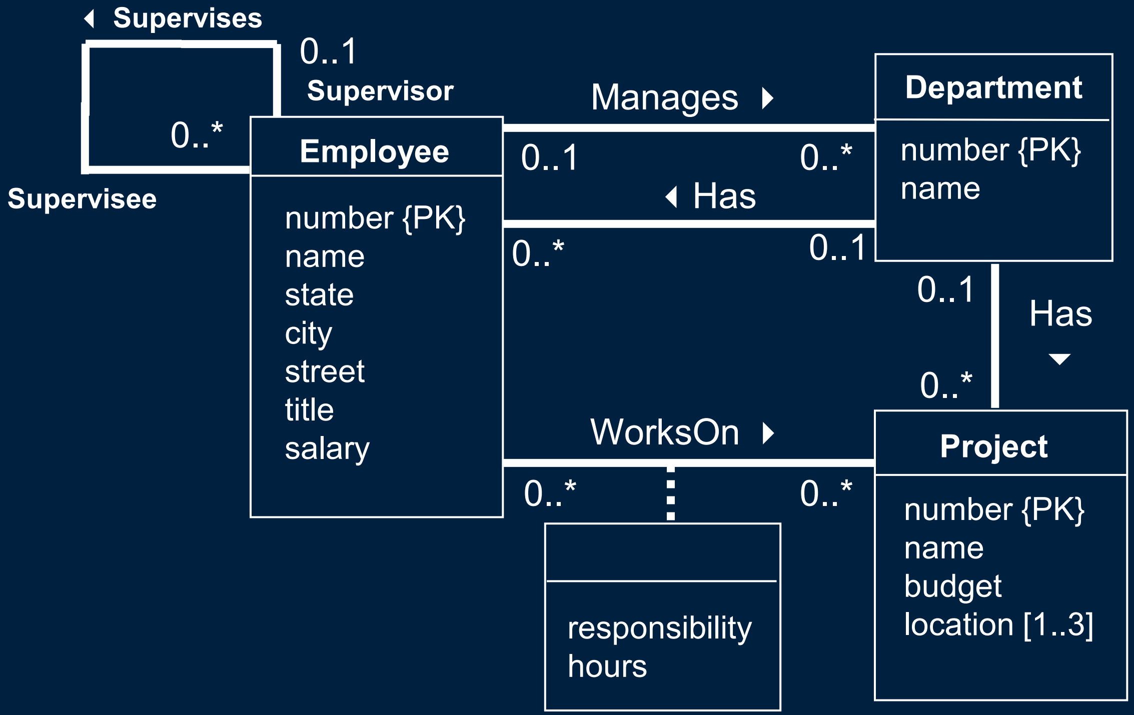
Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Example:

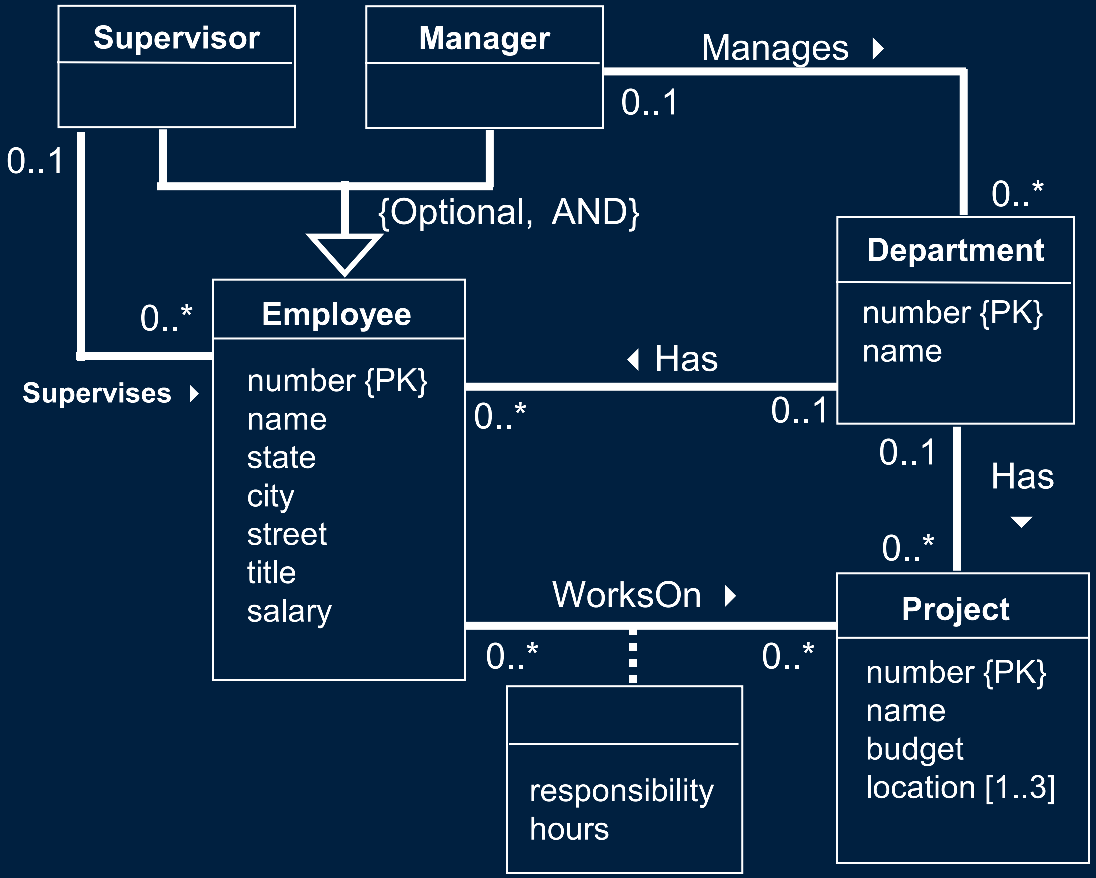


Note: The min-max constraint on the whole side of the relationship (in this case department) must always be 1..1 when modeling composition. Why?

Original ER Model Example



EER Model Example



Conclusion

The ***Enhanced Entity-Relationship*** model (***EER***) model allows for object-oriented design features to be captured.

Generalization and ***specialization*** are two complementary processes for constructing superclasses and subclasses.

Participation and ***disjoint constraints*** apply to subclasses.

- Participation of a superclass may be `mandatory` or `optional` in a subclass.
- A superclass may only be a member of one subclass (disjoint constraint indicated by `OR`) or multiple (indicated by `AND`).

Aggregation and composition are used to model `HAS-A` or `PART-OF` relationships.

The features of EER modeling are rarely needed in most database design projects.

Objectives

Given an EER diagram, recognize the subclasses, superclasses, and constraints using the notation.

Explain the difference between the participation constraint and the disjoint constraint.

Explain the difference between aggregation and composition.

Given an EER diagram, list the attributes of each class including attributes inherited from superclasses.

THE UNIVERSITY OF BRITISH COLUMBIA