Lecture 9 – Data Model Engineering

Open Data Management & the Cloud

(Data Science & Scientific Computing / UniTS – DMG)

Metadata and provenance

- Today data are being collected by a vast number of instruments in every discipline of science

- Datasets currently grow into the petascale range and are shared among and across the scientific communities

- Importance of recording the meaning of data and the way they were produced increases dramatically

- **Metadata**: data descriptions that assign meaning to the data

- Data **provenance**: information about how data was derived

- Both are important to interpret a particular data item

- Metadata catalogs made use of a variety of underlying technologies: relational databases, XML-based databases, grid database services, etc.

- **Modeling** adds value to metadata management much the same way it does for data itself

Why data modeling

- **Quality**: conceptual integrity is the most important consideration in system design

- **Communication**: models reduce misunderstandings and promote consensus among developers, customers, and other stakeholders

- **Reliability**: rigorous modeling improves the quality of the data. You can weave constraints into the fabric of a model and the resulting database

- **Performance**: a sound model simplifies database tuning

Data model (1)

- A model is a representation of some aspect of a problem that lets you thoroughly understand it

- A **data model** is a model that describes how data is stored and accessed

  - It does not include many of the details of how the data is stored or how the operations are implemented

  - It uses logical concepts, such as objects, their properties, and their interrelationships

- Categories of data model

  - **Conceptual data model**: focuses on major entity types and relationship types. Provides a high-level overview. Has no attributes

  - **Logical data model**: fleshes out the conceptual model with attributes and lesser entity types

  - **Physical data model**: converts the logical model into a database design. The emphasis is on physical constructs such as tables, keys, indexes, and constraints.

Data model (2)

- Entity-Relationship (ER) models

  - **Entity**: real-world object or concept

  - **Attribute**: property of interest that further describes the entity

  - **Relationship**: among two or more entities, it represents the associations among the entities

- Additional abstractions for advanced ER models:

  - **Specialization**: Specialization is the process of defining a set of subclasses of an entity type; this entity type is called the superclass of the specialization

    - The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass

  - **Generalization**: reverse process of abstraction in which we identify the common features of several entities, and generalize them into a single superclass of which the original entity types are special subclasses

  - **Categories**: to represent a collection of entities from different entity types

Specialization, Generalization, Category
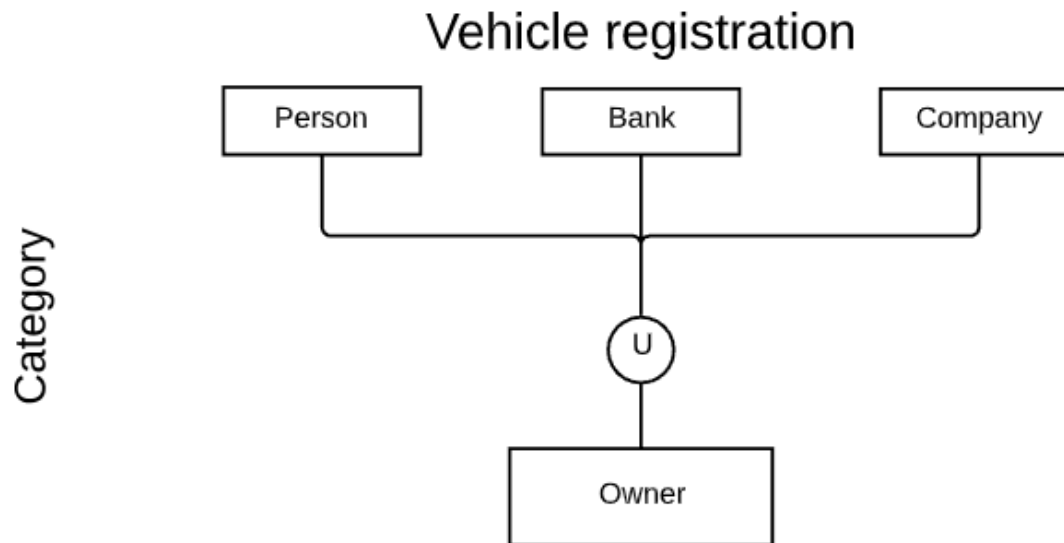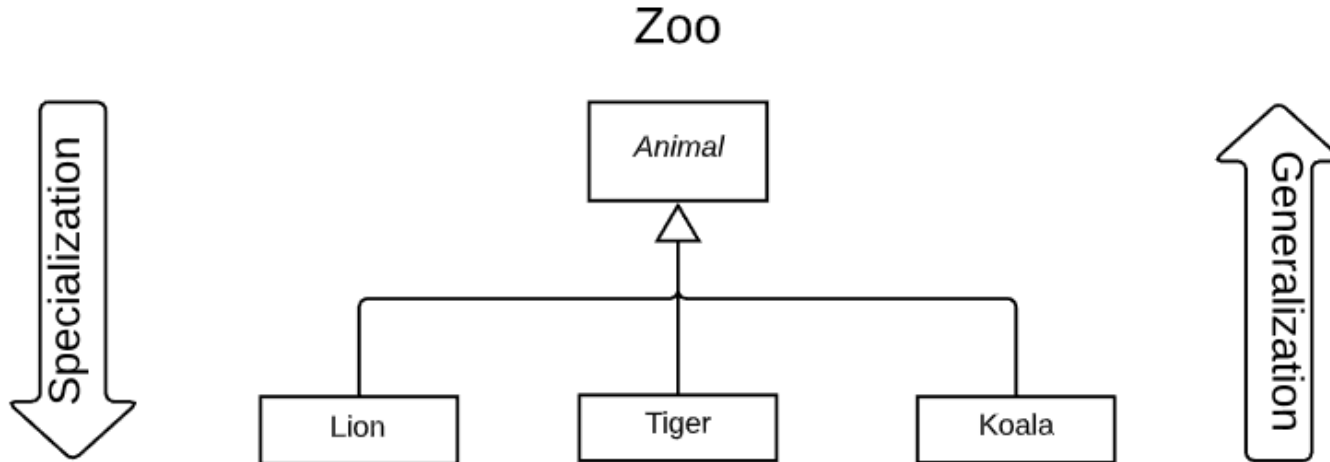
Database systems

- Several criteria can be used to classify Database Management Systems (DMBS)

- One of these criteria is the data model used:

  - **Relational DBMSs**: they use the relational model, which represents a database as a collection of tables, where each table can be stored as a separate file. Most relational databases use the high-level query language called **SQL**

  - **Object databases**: they use object data model, but has not had widespread use

  - **NoSQL databases**:

    - NoSQL key-value stores: simple data model based on fast access by the key to the value associated with the key

    - Document based NoSQL sytems: data in form of documents using well-known formats, such as JSON, accessed by ID or indexes

    - Column-based NoSQL systems: partition a table by column into column families, where each column family is stored in its own files

    - Graph-based NoSQL systems: Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions

  - Hybrid systems: e.g. **XML databases**

Relational database system

- Relational model: represents the database as collection of *relations*

  - Each relation represents a **table** of values, i.e. a flat file of records

  - Each row in the table represents a collection of related data values

  - A row represents a fact that typically corresponds to a real-world **entity** or **relationship**

  - Table name and column names are used to interpret the meaning of the values in each row

- A **relation schema** R, denoted by $R(A_1, A_2, …, A_n)$ is made up of a **relation name** R and a list of **attributes** $A_1, A_2, …, A_n$.

  - The attribute A_i is the name of a role played by some domain D in the relation schema R

  - The degree of the relation R is the number of attributes n

  - D is called domain of A_i and denoted by **dom(A_i)**

  - A **domain** is given: name, **data type** and **format**

Relation (instance)

- A **relation** (or **relation state**) r of a relation schema is a set of n-tuples r = {t_1, t_2, …, t_m} and is denoted as *r(R)*

- Each **n-tuple** t is an ordered list of n values t = <v_1, v_2, …, v_n>, where each value v_i is an element of dom(A_i) or a special NULL value

  - Each value in a tuple is an **atomic** value (not divisible into components)

  - We can have several meanings for NULL values: value unknown, value exists but not available, attribute does not apply (i.e. value undefined)

- A relation *r(R)* is a mathematical relation of degree n on the domains dom(*A_1*), dom(*A_2*), …, dom(*A_n*), which is a subset of the **Cartesian product** of the domains that define R:

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times \cdots \times dom(A_n))$$

- Some relations may represent facts about **entities**, other relations may represent facts about **relationships**

Relational model constraints

- Inherent **model-based constraints** (or implicit constraints)

  - E.g.: a relation cannot have duplicate tuples

- **Schema-based constraints** (or explicit constraints)

  - Typically directly expressed in the schema of the data model, using a Data Definition Language (DDL)

  - E.g.: domain constraints, key constraints (see next slides)

- **Application-based** or semantic **constraints**

  - Constraints that cannot be directly expressed in the schema of the data model

  - They must be enforced by application programs

- For each attribute, a constraint can specify whether NULL values are or are not permitted

Domain and key constraints

- **Domain constraints** specify that each value of each attribute A must be an atomic value from the domain dom(A)

  - Data types associated with domains typically include standard numeric types for integers, real numbers, characters, booleans, fixed-length and variable length strings, date, time, etc.

- A subset of attributes of the relation schema R is called **superkey** (SK) of R if for any two distinct tuples t_1 and t_2 of a relation state r of R, $t_1[SK] \neq t_2[SK]$

  - Every relation has at least one default SK, the set of all its attributes

- A **candidate key** (CK) of a relation schema R is a SK of R with the additional property that removing any attribute from CK leaves a set of attributes that is not a superkey of R

  - A relation schema may have more than one CK

- A **primary key** (PK) is a CK whose values are used to *identify* tuples in the relation

  - It is usually better to choose a PK with a single attribute or a small number of attributes

  - A PK composed of one column is called single primary key, a combination of column is called composite primary key

  - The other candidate keys are designated as **unique keys** (or **alternate keys**)

Relational database

- A relational database usually contains many relations

- A **relational database schema** S is a set of relation schemas S = {R_1, R_2, …, R_m} and a set of **integrity constraints** (IC)

- A relational database **state** DB of S is a set of relation states DB = {r_1, r_2, …, r_m} such that r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC

- A database state that does not obey all the integrity constraints is called an **invalid state**, and a state that satisfy all the constraints in the defined set of integrity constraints IC is called a **valid state**

- Each relational DBMS must have a **data definition language** (DDL) for defining a relational database schema

  - Current relational DBMS-s are mostly using the SQL language for this purpose

Entity and referential integrity constraints

- The **entity integrity constraint** states that no primary key value can be NULL

  - The primary key is used to identify individual tuples in the relation

- A **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples of the two relations

  - A set of attributes FK in relation schema R_1 is a **foreign key** of R_1 that **references** relation R_2 if it satisfies the following rules:

    1. The attributes in FK have the same domain(s) as the primary key attributes PK of R_2.

    2. A value of FK in a tuple t_1 in the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL. In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 references the tuple t_2

  - If the two conditions above hold between R_1 and R_2, a **referential integrity constraint** from R_1 to R_2 is said to hold

Relational model example

```
-- SQLite tips

PRAGMA foreign_keys = ON;
.mode column
.header on
```

```
sqlite> SELECT * FROM course;
id          name                cfu
----------  ------------------  -----
0           Open Data Management  6
1           Machine Learning      8
```

```
sqlite> SELECT * FROM exam;
student_id  course_id  grade
----------  ---------  ----------
0           1          25
1           0          28
1           1          30
```

```
CREATE TABLE student(
id          INTEGER PRIMARY KEY AUTOINCREMENT,
first_name VARCHAR(20) NOT NULL,
mid_name    VARCHAR(20),
last_name   VARCHAR(20) NOT NULL,
birth_day   DATE NOT NULL,
email       VARCHAR(20) UNIQUE NOT NULL);

CREATE TABLE course(
id    INTEGER PRIMARY KEY AUTOINCREMENT,
name VARCHAR(20) UNIQUE NOT NULL,
cfu   INTEGER NOT NULL);

CREATE TABLE exam(
student_id INTEGER,
course_id   INTEGER,
grade       INTEGER NOT NULL,
FOREIGN KEY(student_id) REFERENCES student(id),
FOREIGN KEY(course_id) REFERENCES course(id));
```

```
sqlite> SELECT * FROM student;
id          first_name  mid_name    last_name   birth_day   email
----------  ----------  ----------  ----------  ----------  --------------
0           Luca                    Rossi       1990-12-01  rossi@email.com
1           Luca                    Rossi       1985-05-05  rossi@email.it
2           Maria       Grazia      Bianchi     1985-05-05  maria@email.com
```

Relational model operations

- The operations of the relational model can be categorized into *retrievals* and *updates*

- There are three basic operations that can change the states of relations in the database: Insert, Delete, and Update (or Modify)

  - Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated

  - The **Insert** operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. It can violate any of the four type of constraints (domain constraint, key constraint, entity integrity and referential integrity)

  - The **Delete** operation can violate only the referential integrity

  - The **Update** (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R

    - It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified

    - Updating an attribute that is neither part of a primary key nor part of a foreign key usually causes no problems

The Transaction concept

- A database application program running against a relational database typically executes one or more *transactions*

- A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database

- At the end of the transaction, it must leave the database in a **valid or consistent state** that satisfies all the constraints specified on the database schema

- A single transaction may involve any number of retrieval operations and any number of update operations

- A large number of commercial applications running against relational databases in online transaction processing (**OLTP**) systems are executing transactions at rates that reach several hundred per second

Entity-relationship diagram's

- Many different Entity-Relationship Diagrams (ERD) notations available

  - Chen notation, **Information Engineering** (**IE**) or Crows's foot notation, IDEF1X, **Unified Modeling Language (UML)**, etc.

- The Information Engineering is a modeling notation that has been in use for many years

- IE focuses on details such as tables, keys, and indexes (it is closer to the Physical data model). IE's attention to database detail is helpful for explaining nuances of the UML

- The IE lacks a standard notation and there are several variants

- The UML class model specifies classes (entity types) and their relationship types. It is closer to the Conceptual data model:

  - More concise than traditional database notations (usually no keys, foreign keys, indexes and referential integrity)

  - It provides an higher level of abstraction

Data model design: UML approach

- Any database with a schema that includes more than 20 entity types and a similar number of relationship types requires a careful design methodology

    - Covering requirements analysis, modeling, design, implementation and deployment of databases

- One approach is a standard proposed by the Object Management Group (OMG): the Unified Modeling Language (UML)

    - A diagrammatic notation and associated language syntax to cover the entire software life cycle

    - Can be used by software developers, data modelers, database designers

    - Language-independent and platform-independent

    - It specifies many types of diagrams for various software design purposes

UML class diagram

- The UML class diagram specifies classes (entity types) and their relationship types

- An *object* is a concept, abstraction, or thing that has identity and meaning for an application

  - Application needs also determine the level of abstraction for representing an object

  - E.g.: an airplane flight can be represented by departure/arrival time or as a sequence of phases (at gate, boarding, taking off, en route, landing, at gate, disembarking) depending on the applications

- A **class** describes a group of objects with similar properties (attributes), behavior (operations), relationships to other objects, and semantic intent

Classes and attributes

UML   **IE**
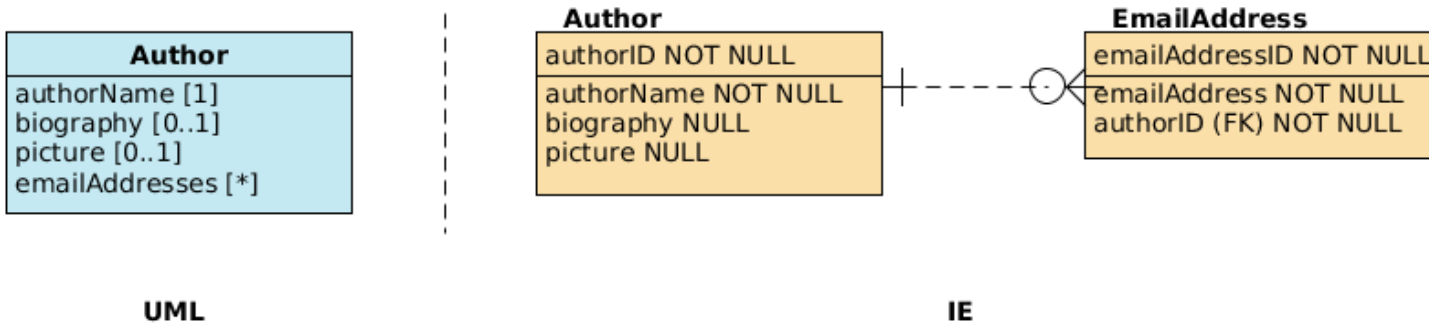
- An **attribute** is a named property of a class that describes a value held by each object of the class

  - The second portion of the UML class box shows attribute names

- The IE notation lists attributes in both portions of the entity type box.

  - The top portion has primary key attributes, the lower portion has the remaining data attributes

  - The attribute authorID above is a surrogate key (a generated number that uniquely identifies an author)

- In UML, each attribute **can have** an attribute multiplicity that specifies the number of possible values for each record. If not specified, it defaults to [1].

- Normally, a relational database attribute cannot store a collection of values

  - For IE, we had to convert the "many" multiplicity to a relationship type

Data types

Order	
orderNumber: varchar(20)	
orderDateTime: datetime	
shippingMethod: varchar(20)[0..1]	
taxAmount: decimal(18,2)	
shippingHandlingAmount: decimal(18,2)	
totalAmount: decimal(18,2)	

UML

Order
orderID NOT NULL
orderNumber: orderNumber NOT NULL
orderDateTime: datetime NOT NULL
shippingMethod: shippingMethod NULL
taxAmount: money NOT NULL
shippingHandlingAmount: money NOT NULL
totalAmount: money NOT NULL

IE

- It is good database practice for developers to assign each attribute a domain (IE) and then separately resolve the domain to a data type

  - Flexibility: there are fewer domains than attributes

  - A domain can define both a data type and additional constraints

- But most UML tools just assign each attribute a data type

- The UML notation lists the attribute name, a colon, the data type, and attribute multiplicity

- The IE notation lists the attribute name, a colon, the domain (optional), the data type (optional, can appear with or without the domain), and nullability

Class operations

- An **operation** is a function or procedure that can be applied to or by objects in a class.
  IE models lack operations.

- The third portion of the UML class box displays operation names

- A UML operation summarizes business logic. It is helpful to see a summary of functionality placed in context with the model of data structure

  - It helps designing the data model itself

- Database stored procedures can implement operations
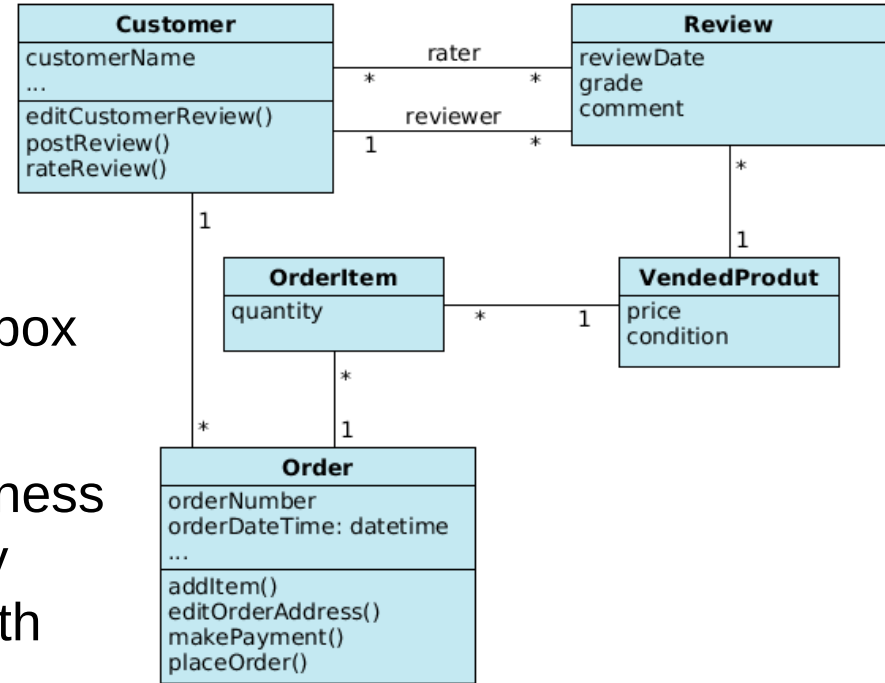
Notation summary (1)

UML Concept	UML Notation	IE Concept	IE Notation	Definition
Object		Entity		A concept, abstraction, or thing that has identity and meaning for an application.
Class	▭	Entity type	▭	A group of objects with similar attributes, behavior, relationships to other objects, and semantic intent.
Value		Value		A piece of data that lacks identity.
Attribute	▤	Attribute	▤ ⬭	A named property of a class that describes a value held by each object of the class.
Operation	▤			A function or procedure that can be applied to or by objects in a class.
		Domain		The named set of possible values for an attribute,
Data type		Data type		A specification of type and size for values such as long integer, varchar(20), and date.

Associations

- **Associations** provide the means for relating classes

- The UML notation for an association is a line

- A UML association corresponds to an IE **identifying relationship type** (solid line) and **non-identifying relationship type** (dashed line)
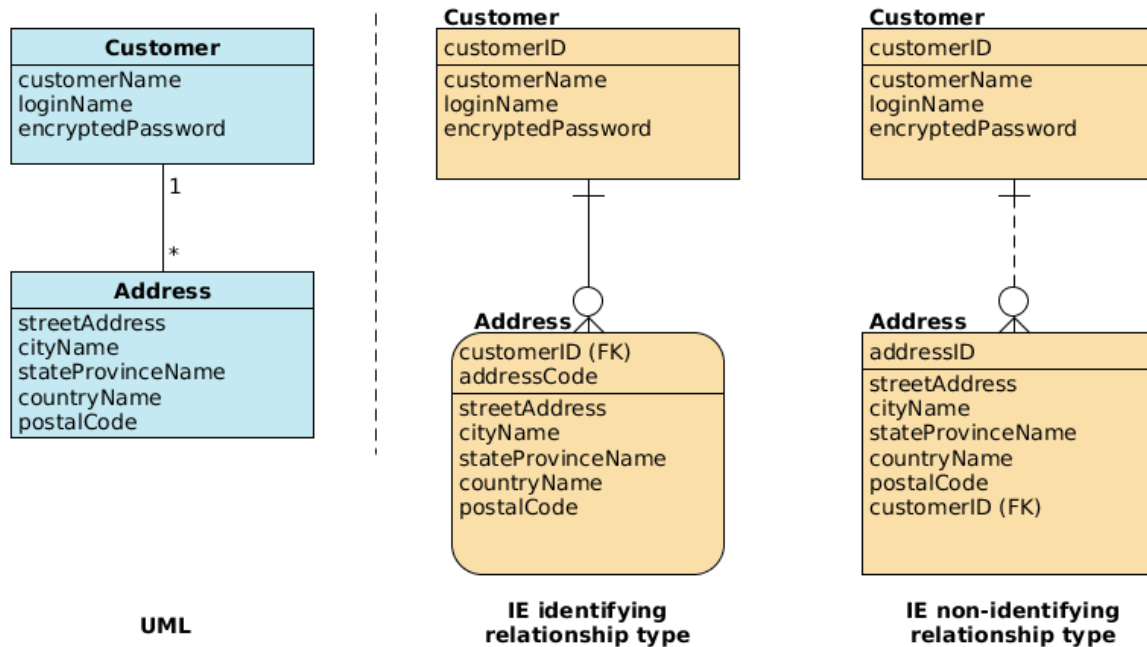
Independent and dependent entity types

- IE distinguishes between **identifying relationship type** (solid line) and **non-identifying relationship type** (dashed line)

  - An identifying relationship type propagates primary key attributes of the source entity type to the primary key of the referent entity type. A solid line connects the entity types. The referent entity type is necessarily dependent (rounded box).

  - A non-identifying relationship type propagates primary key attributes of the source entity type to data attributes of the referent entity type. A dashed line connects the entity types. The referent entity type may be independent (square box) or dependent (rounded box) depending on its other relationship types and generalizations (next slides).

- IE distinguishes between **independent entity types** (square box) and **dependent entity types** (rounded box)

  - An independent entity type (also called **strong entity type**) does not include any foreign keys in its primary key. The IE symbol is a square-corner box

  - A dependent entity type (also called **weak entity type**) includes one or more foreign keys in its primary key (via one or more identifying relationship types or via generalization, see the next slides). It can exist only if one ore more other entity types also exist. The IE symbol is a rounded-corner box

Multiplicity

- **Multiplicity** specifies the number of occurrences of one class that may relate to a single occurrence of an associated class

- Thus multiplicity pertains to an association end

Multiplicity	Option	Cardinality
0..0	**0**	Collection must be empty
0..1		No instances or one instance
1..1	**1**	Exactly one instance
0..*	*****	Zero or more instances
1..*		At least one instance
5..5	**5**	Exactly 5 instances
m..n		At least m but no more than n instances

UML multiplicity

IE relationship symbols

| Notation	Meaning
	Relationship
	One
	Many
	One and only one
	Zero or one
	One or many
	Zero or many
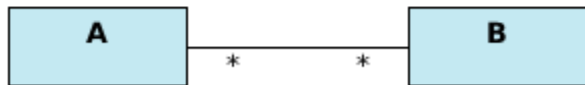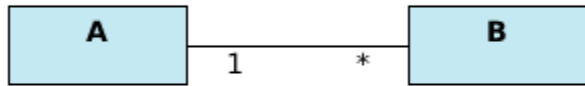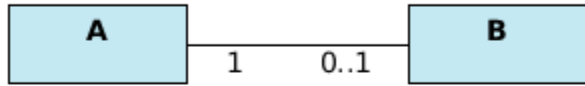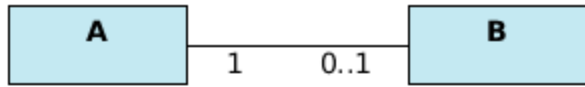
UML
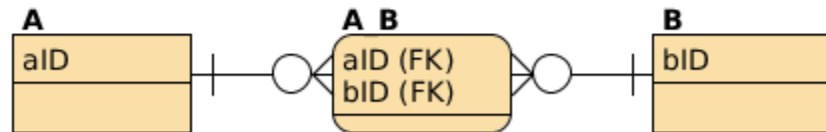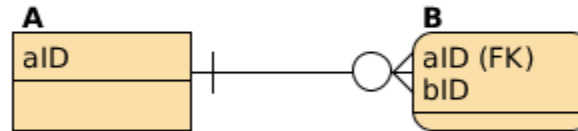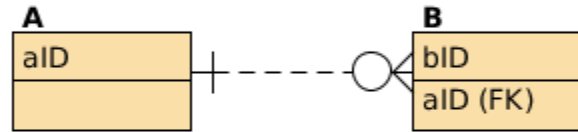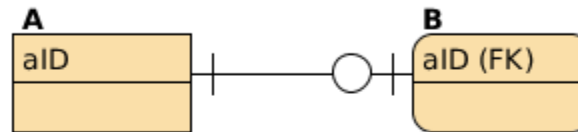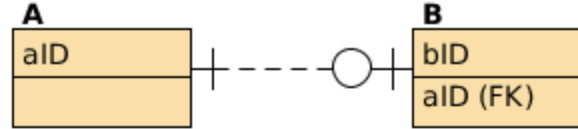
IE

Many-to-many relationships
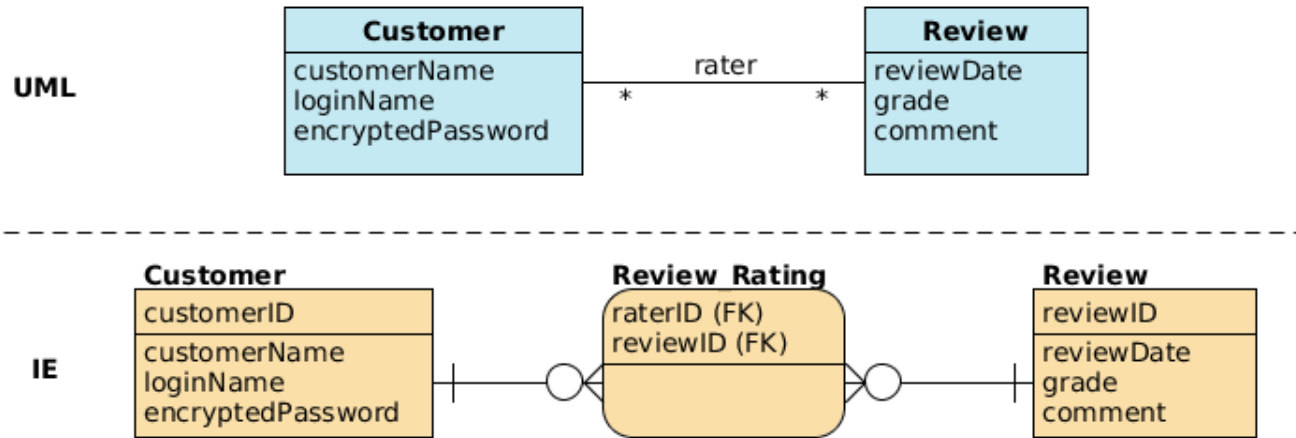
- This example shows a **many to many relationship** between Customer and Review, i.e. a customer can rate 0 or more reviews and a review can be rated by 0 or more customers

- The physical data model of the IE notation shows that this type of relation is realized with a dependent entity type (Review_Rating) and two or more identifying relationship types

- Review_Rating is called an **associative entity type**, i.e. it obtains its primary key from two or more entity types.

  - Review_Rating.raterID refers to Customer.customerID

Association names (UML)

- The UML only requires **association names** when there are multiple associations between the same classes

- An association name often reads in a particular direction. Nevertheless, associations can be traversed either way

  - The UML also has a navigation icon to show the direction for reading the name

- This association traversal is analogous to combining relational database tables via foreign-key-to-primary-key joins

- An **association end name** is an alias for a class in an association. The UML notation is a legend next to the class-association intersection

  - Association end names are optional if a model is unambiguous

  - Ambiguity occurs when there are multiple associations for the same classes or an association for objects of the same class

- When constructing models, you should properly use association ends and not introduce a separate class for each reference

Association names example
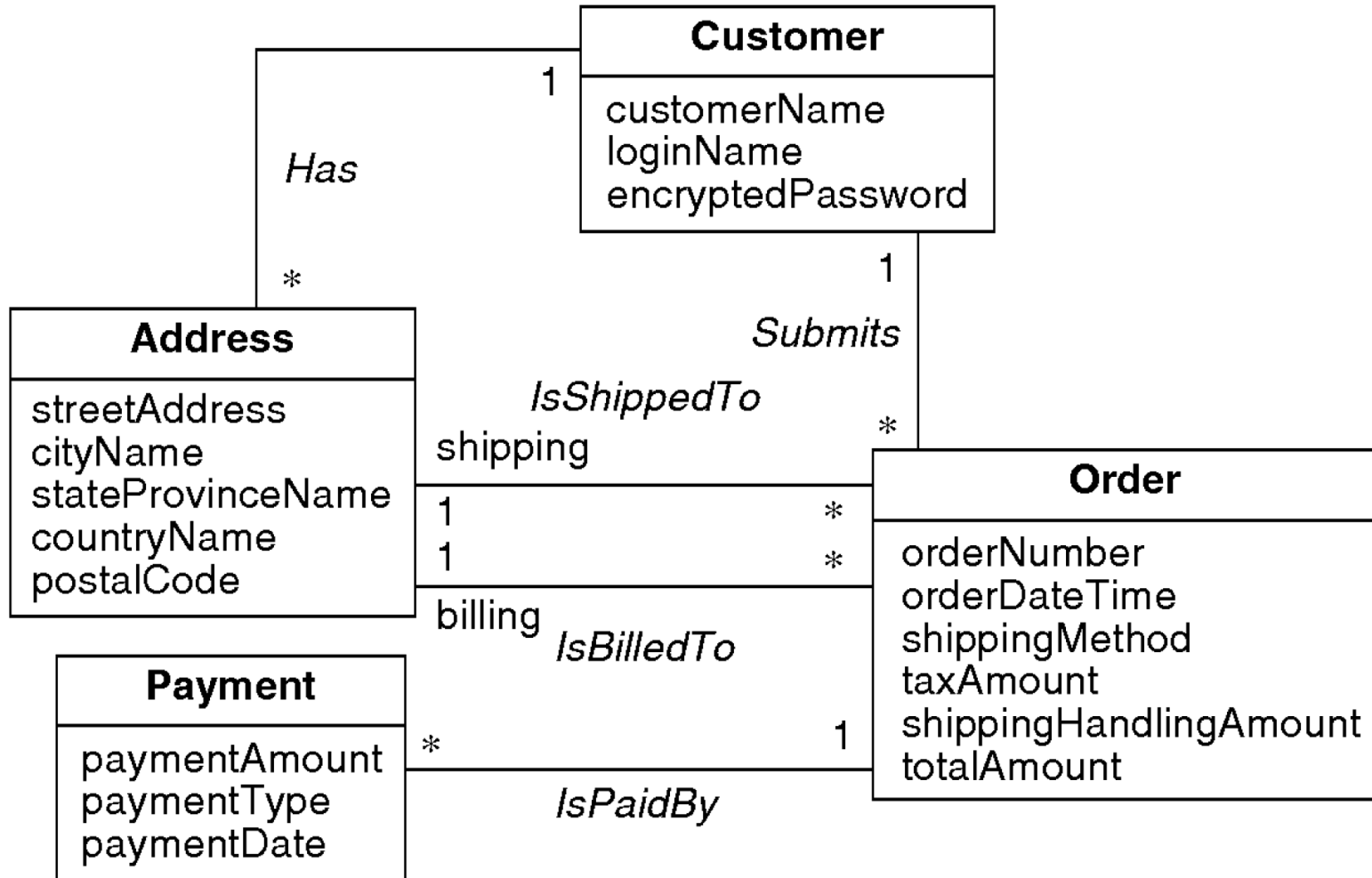
# Benefits of association names

- Benefits of association names:

  - Improves model readability

  - Provides a table name for an associative entity type

  - Disambiguates multiple associations for the same classes

- Benefits of association end names:

  - Improves model readability

  - Provides a foreign key name

  - Disambiguates multiple associations for the same classes

  - Disambiguates an association for objects of the same class

  - Provides clarity for model traversal and SQL queries

Relationship names (IE)

- It is a common IE practice to include relationship type names. Each relationship type can have either a single name or a pair of directed names. Directed names add bulk but make a model more readable.

- A single name can be useful for development (it provides a table name)
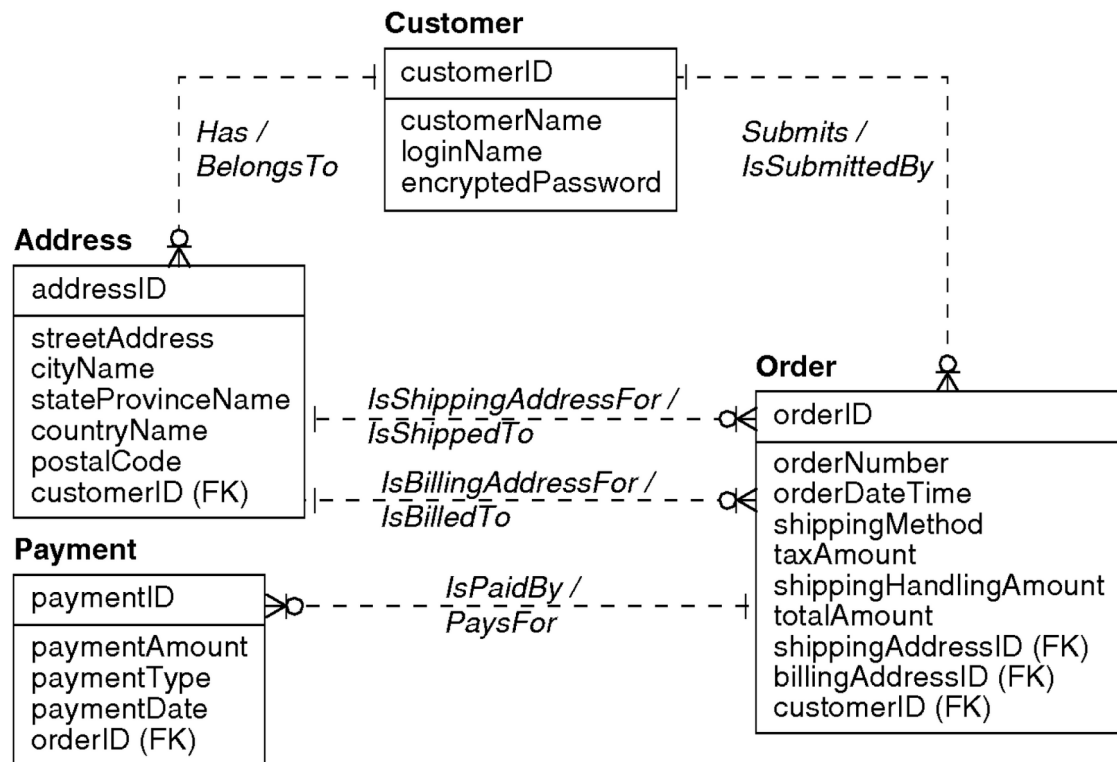
Notation summary (2)

UML Concept	UML Notation	IE Concept	IE Notation	Definition
Link		Relationship		A physical or conceptual connection among objects.
Association	———	Relationship type	- - - - - ———	Describes a group of links with common structure and semantics.
Association end		Role		The use of a class in an association.
		Associative entity type	+⊣◀◻▷⊢+	Resolves a many-to-many relationship type.
Multiplicity	1 0..1 *	Cardinality (though technically incorrect)	⏐ d d⋖	Specifies the number of occurrences of one class that may relate to a single occurrence of an associated class.
		Identifying relationship type	———	Propagates source primary key attributes to the referent primary key.
		Non-identifying relationship type	- - - - -	Propagates source primary key attributes to referent data attributes.
		Independent entity type	▭	Has no foreign keys in its primary key. Also called a strong entity type.
		Dependent entity type	⬭	Includes foreign key(s) in its primary key. Also called a weak entity type.

Generalization

- Generalization is a defining characteristic of object-oriented software approaches and organizes classes by their similarities and differences

  - It leads to smaller models with deeper insight

- Generalization couples a class (the **superclass**) to one or more variations of the class (the **subclasses**)

- The superclass holds common information (attributes, operations, and associations)

- Each subclass adds specific information

- Generalization organizes classes by their similarities and differences, structuring the description of objects. Generalization can arise from requirements that list structural alternatives

- The UML notation for generalization is a large hollow arrowhead that points to the superclass.

Example of inheritance in UML
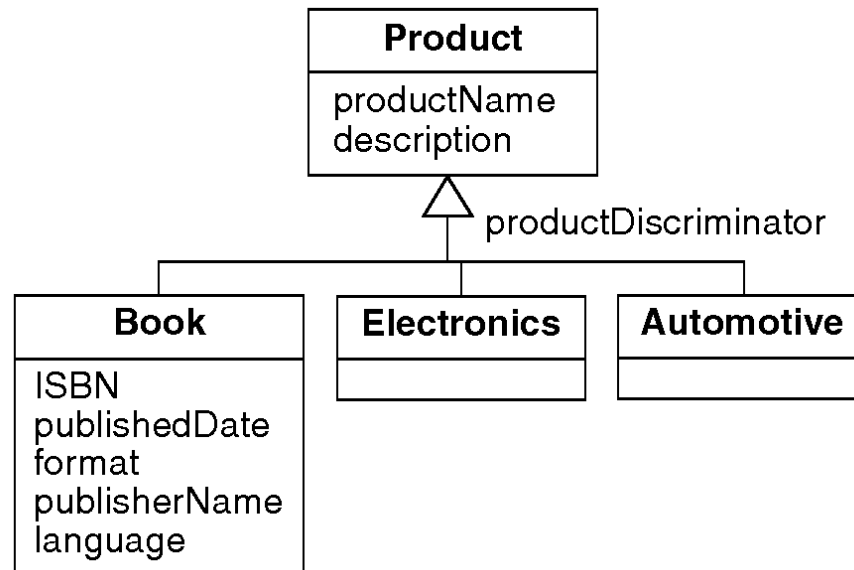
- The generalization set name (productDiscriminator) is an enumerated attribute that can be placed next to the generalization symbol

- Generalization has two purposes

  - Reuse. Subclasses can share information that superclasses provide

  - Form a taxonomy and declare what is similar and what is different about classes. This is much more profound than modeling each class individually and in isolation

Abstract class

- An **abstract class** is a class that has no direct occurrences. The UML indicates an abstract class by italicizing the class name or placing the legend {abstract} before or after the class name

- A superclass can be abstract or concrete, depending on how the generalization is stated

- As a matter of style, it is a good idea to avoid concrete superclasses. Then, abstract and concrete classes are readily apparent at a glance; all superclasses are abstract and all leaf subclasses are concrete.

- Deeply nested generalizations, try to avoid generalizations with more than five levels

Nested generalization
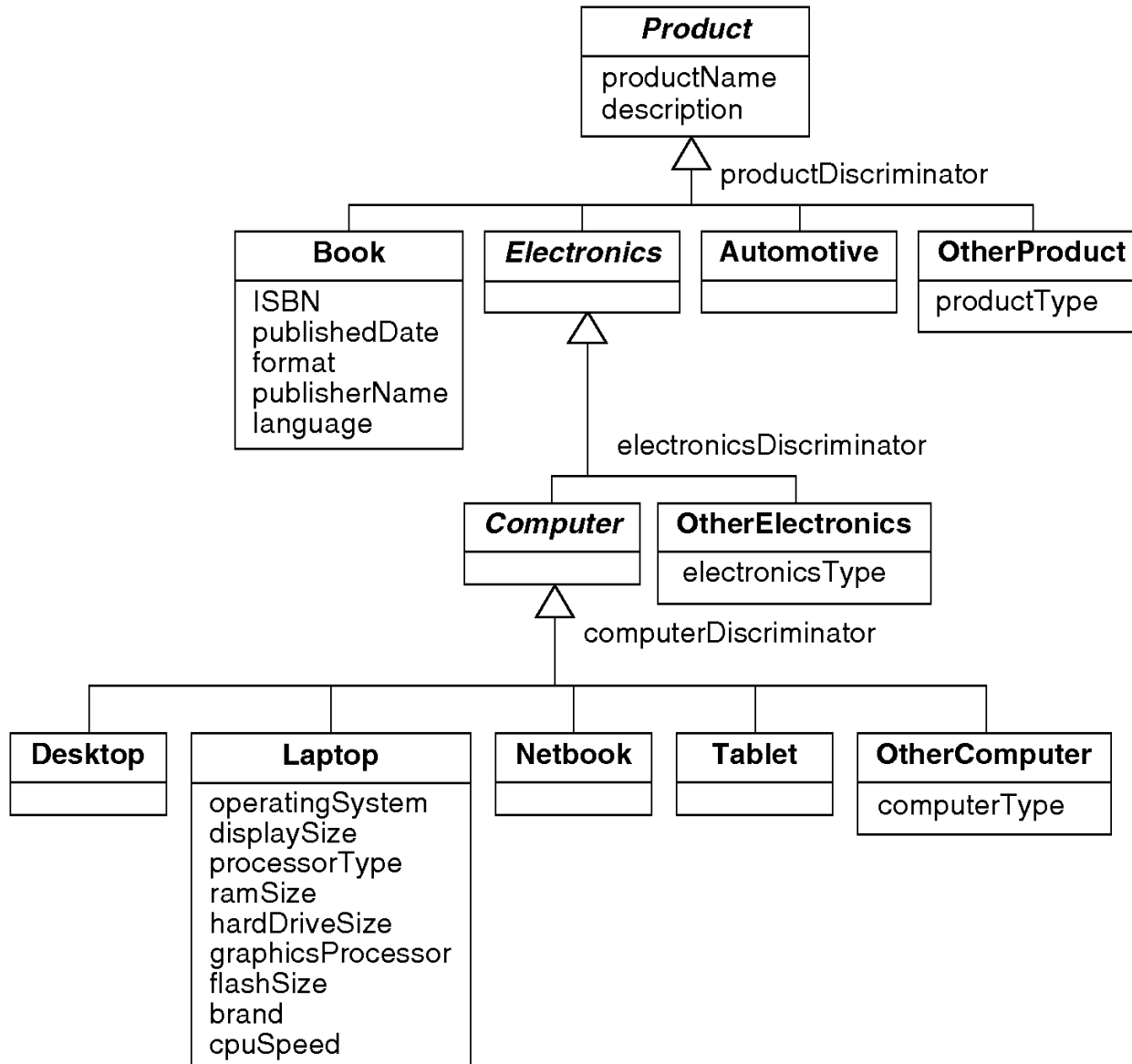
IE notation for generalization

- IE subtypes are dependent entity types because each subtype primary key refers to the supertype primary key

- The supertype may be independent or dependent (but is usually independent) based on whether its primary key incorporates a foreign key from another entity type.
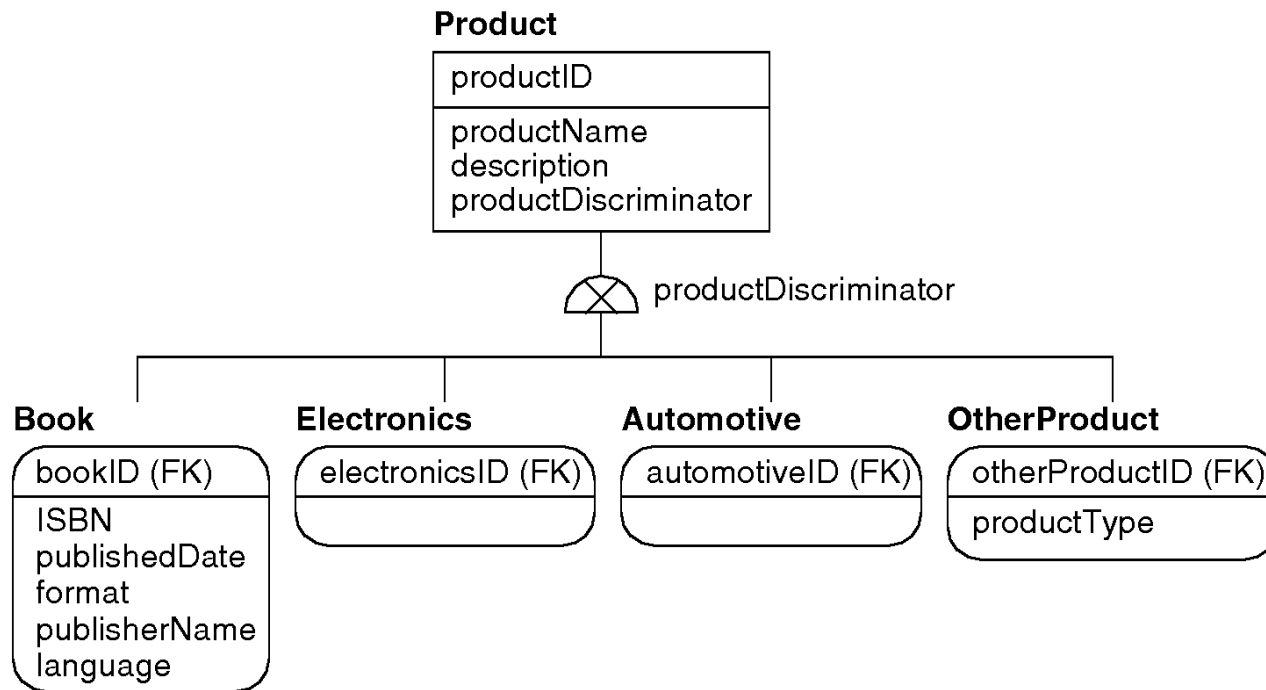
UML Concept	UML Notation	IE Concept	IE Notation	Definition
Generalization		Subtyping		An organization of classes by their similarities and differences, structuring the description of objects.
Superclass		Supertype		The common attributes, operations, and associations for a generalization.
Subclass		Subtype		Specific attributes, operations, and associations for a generalization.
Generalization set name		Discriminator		An enumerated attribute that indicates the subclass that applies for each superclass occurrence.
Abstract class				A class with no direct occurrences.
Concrete class				A class that can have direct occurrences.
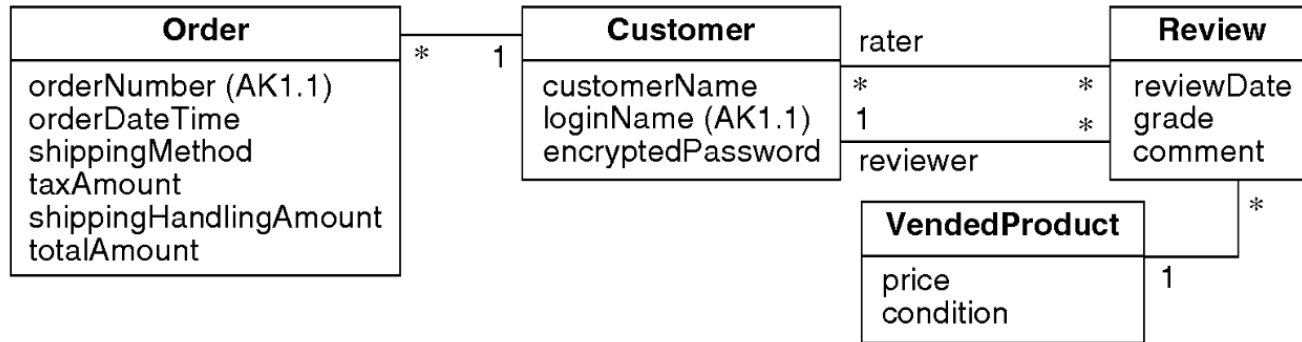
- An **alternate key** is a candidate key that is not chosen as a primary key. Therefore each candidate key is either a primary key or an alternate key

- The UML has no specified notation for unique keys (i.e. alternate keys)

- It is possible to use the same notation used by IE, the **AKn.m** notation (see figure above)

  - AKn.m = column m^{th} of the nth Alternate Key

Surrogate key vs Natural key (1)

- With **existence-based identity** each class has a generated identifier (also called a **surrogate key**) as its primary key. Each association has a primary key composed of identifiers from the related classes

  - The advantage of this approach is that each class's primary key is a single attribute (often defined as a number)

  - Furthermore, since the primary key is synthetic, it is immutable

- Another approach is **value-based identity** — a unique combination of real-world attributes (also called a **natural key**) identifies each class occurrence. "Real-world attributes" are those that come from the business problem description

  - A downside is that the value of real-world attributes can change — such changes must propagate to foreign keys

  - Some models have a series of dependent entity types that lead to unwieldy multi-attribute primary keys

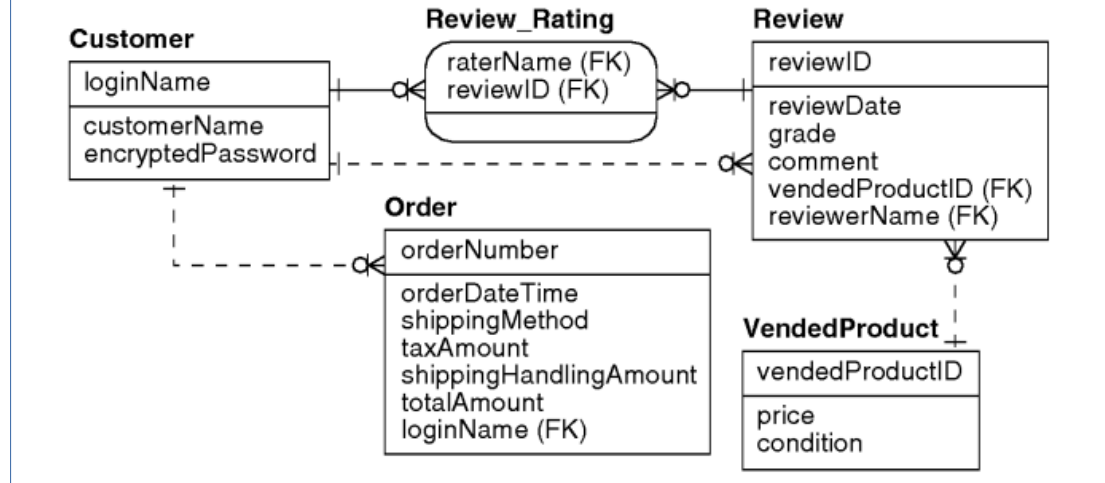- Unless there are unusual circumstances, it is recommend the use of surrogate keys (existence-based identity).

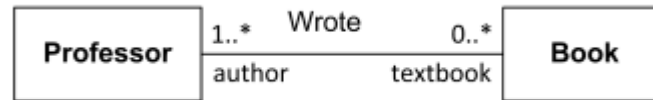# Surrogate key vs Natural key (2)

Surrogate Key Example
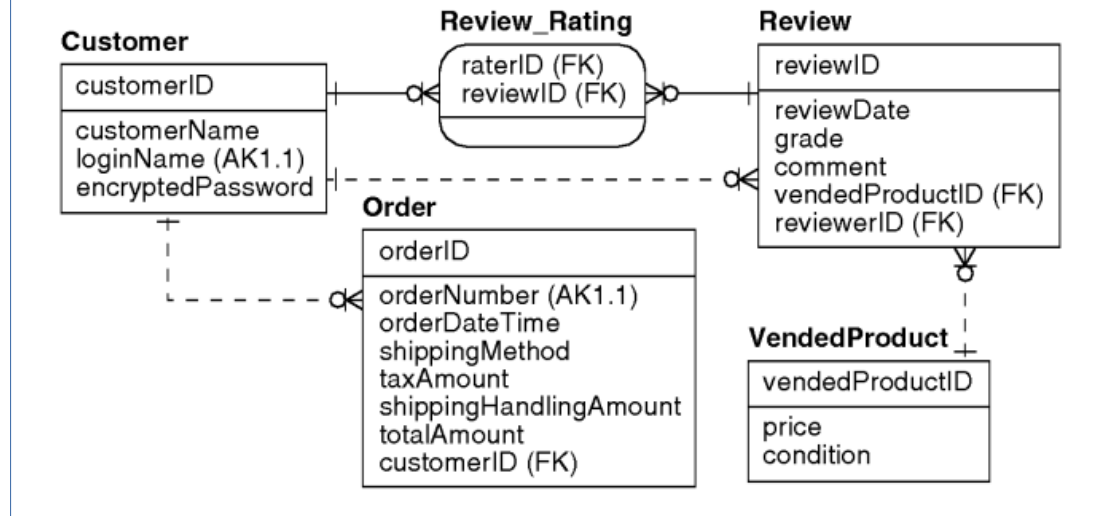
Natural Key Example

# Association, Aggregation, Composition

- Association is a structural relationship that represents objects can be connected or associated with another object inside the system

- Aggregation and Composition are subsets of Association. In both object of one class "owns" object of another class:

  - Aggregation implies a relationship where the child can exist independently of the parent. Example: Course (parent) and Student (child). Delete the Course and the Students still exist.
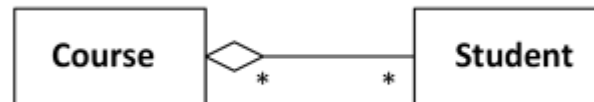

  - Composition implies a relationship where the child cannot exist independent of the parent. Example: Hospital (parent) and Department (child). Departments don't exist separate to a Hospital.
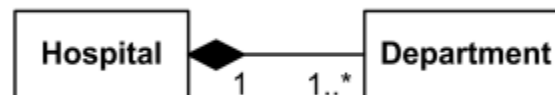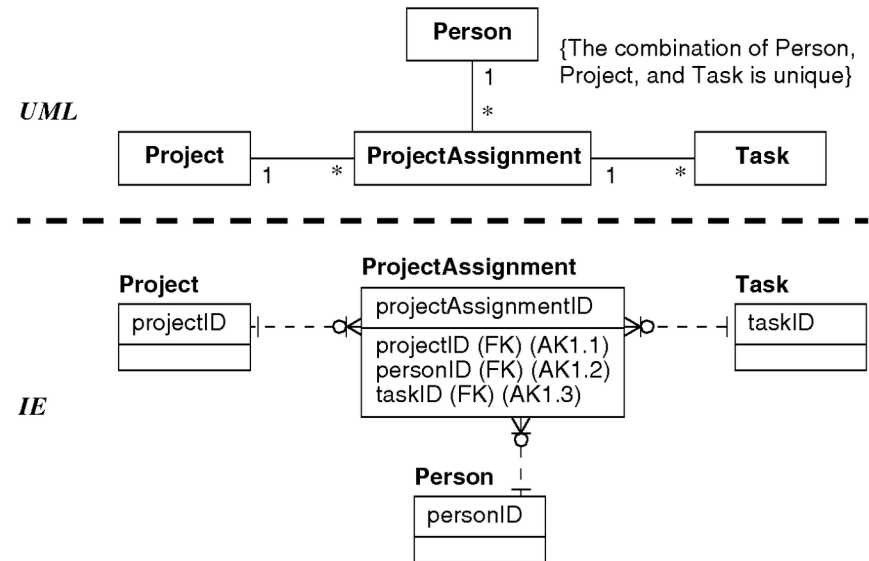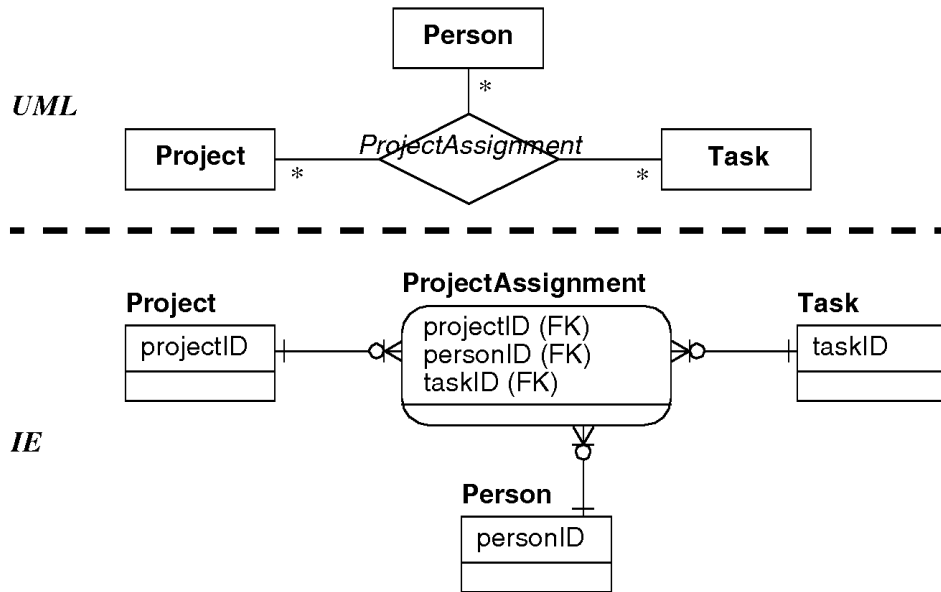
Association class

- An **association class** is an association that is also a class. Like the links of an association, the occurrences of an association class derive identity from the related objects

- Like a class, an association class can have attributes, operations, and associations

- The UML notation for an association class is a box that connects to the corresponding association with a dotted line
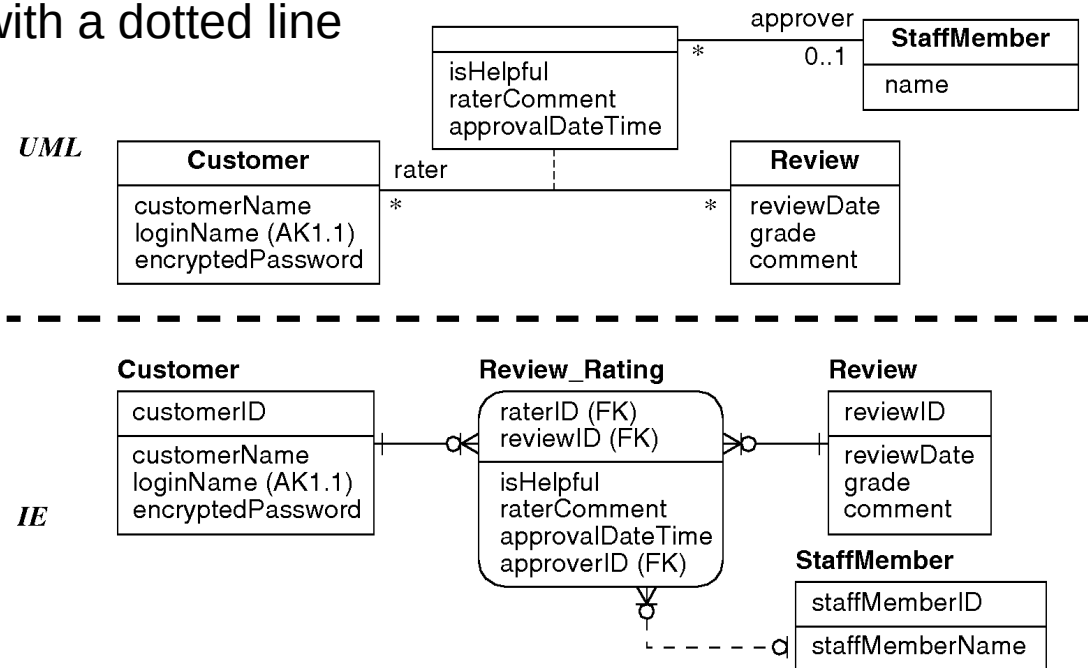
Ternary associations

- A **ternary association** is an association involving three classes

- The UML notation is a diamond with lines connecting the related classes

- Many supposed "ternary" associations are not fundamental and can be decomposed into binary associations, with possible qualifiers and attributes

References

- Blaha, Michael. (2013). UML Database Modeling Workbook

- https://www.yworks.com/products/yed

- https://www.lucidchart.com

- https://www.youtube.com/watch?v=UI6lqHOVHic

- https://www.uml-diagrams.org

- https://www.guru99.com/uml-diagrams.html