Conceptual Data Models for Database Design

Entity Relationship (ER) Model

The most popular high-level conceptual data model is the ER model. It is frequently used for the conceptual design of database applications.

Entity	**Attributes**	**Values**
Car	Color	Red
	Make	Volkswagen
	Model	Bora
	Year	2000

The diagrammatic notation associated with the ER model, is referred to as the ER diagram. ER diagrams show the basic data structures and constraints.

Entity Types, Entity Sets, Attributes and Keys

- The basic object of an ER diagram is the entity. An entity represents a 'thing' in the real world.
- Examples of entities might be a physical entity, such as a student, a house, a product etc, or conceptual entities such as a company, a job position, a course, etc.
- Entities have attributes, which basically are the properties/characteristics of a particular entity.

Examples of entities and attributes:

There are several types of entities. Including:
- Simple vs. Composite
- Single-valued vs. Multi-valued
- Stored vs. Derived

Simple vs. Composite Attributes

- Composite attributes can be divided into smaller subparts, which represent more basic attributes, which have their own meanings.

- A common example of a composite attribute is Address. Address can be broken down into a number of subparts, such as Street Address, City, Postal Code. Street Address may be further broken down by Number, Street Name and Apartment/Unit number.

- Attributes that are not divisible into subparts are called simple or atomic attributes.

- Composite attributes can be used if the attribute is referred to as the whole, and the atomic attributes are not referred to. For example, if you wish to store the Company

Location, unless you will use the atomic information such as Postal Code, or City separately from the other Location information (Street Address etc) then there is no need to subdivide it into its component attributes, and the whole Location can be designated as a simple attribute.

- What are examples of other composite attributes?

Single-Valued vs. Multi-valued Attributes

- Most attributes have a single value for each entity, such as a car only has one model, a student has only one ID number, an employee has only one data of birth. These attributes are called single-valued attributes.
- Sometimes an attribute can have multiple values for a single entity, for example, a doctor may have more than one specialty (or may have only one specialty), a customer may have more than one mobile phone number, or they may not have one at all. These attributes are called multi-valued attributes.
- Multi-valued attributes may have a lower and upper bounds to constrain the number of values allowed. For example, a doctor must have at least one specialty, but no more than 3 specialties.

Stored vs. Derived Attributes

- If an attribute can be calculated using the value of another attribute, they are called derived attributes.
- The attribute that is used to derive the attribute is called a stored attribute.
- Derived attributes are not stored in the file, but can be derived when needed from the stored attributes.

Null Valued Attributes

- There are cases where an attribute does not have an applicable value for an attribute. For these situations, the value null is created.
- A person who does not have a mobile phone would have null stored at the value for the Mobile Phone Number attribute.
- Null can also be used in situations where the attribute value is unknown. There are two cases where this can occur, one where it is known that the attribute is valued, but the value is missing, for example hair color. Every person has a hair color, but the information may be missing. Another situation is if mobile phone number is null, it is not known if the person does not have a mobile phone or if that information is just missing.

Complex Attributes

- Complex attributes are attributes that are nested in an arbitrary way.
- For example a person can have more than one residence, and each residence can have more than one phone, therefore it is a complex attribute that can be represented as:
- {Multi-valued attributes are displayed between braces}
- (Complex Attributes are represented using parentheses)

e.g.
{**AddressPhone**({*Phone*(AreaCode, PhoneNumber)}, *Address*(StreetAddress(Number, Street, ApartmentNumber), *City, State, Zip*))}

Entity Types, Entity Sets, Keys and Value Sets

Entity Types and Entity Sets

- An **entity type** defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attributes. The entity share the same attributes, but each entity has its own value for each attribute.

Entity Type Example:

- *Entity Type*:
  Student
- *Entity Attributes***:**
  StudentID,
  Name,
  Surname,
  Date of Birth,
  Department
- The collection of all entities of a particular entity type in the database at any point in time is called an entity set. The entity type (Student) and the entity set (Student) can be referred to using the same name.

Entity Set Example:

- Entity Type: Student
- Entity Set:
  [123, John, Smith, 12/01/1981, Computer Technology]
  [456, Jane, Doe, 05/02/1979, Mathematics]
  [789, Semra, Aykan, 02/08/1980, Linguistics]

The entity type describes the **intension**, or schema for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into the entity set, called the **extension**.

Key Attributes of an Entity Type

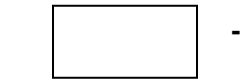
- An important constraint on entities of an entity type is the uniqueness constraint.
- A key attribute is an attribute whose values are distinct for each individual entity in the entity set.
- The values of the key attribute can be used to identify each entity uniquely.
- Sometimes a key can consist of several attributes together, where the combination of attributes is unique for a given entity. This is called a composite key.
- Composite keys should be minimal, meaning that all attributes must be included to have the uniqueness property.
- Specifying that an attribute is a key of an entity type means that the uniqueness property must hold true for every entity set of the entity type.
- An entity can have more than one key attribute, and some entities may have no key attribute. Those entities with no key attribute are called weak entity types.

Value Sets (Domains) of Attributes

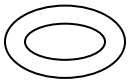
- Each simple attribute of an entity is associates with a domain of values, or value set, which specifies the set of values that may be assigned to that attribute for each entity. For example, date of birth must be before today's date, and after 01/01/1900, or the Student Name attribute must be a string of alphabetic characters.

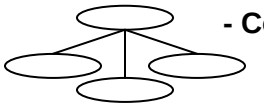
- Value sets are not specified in ER diagrams.

ER Diagram Notation


- Entity


- Attribute


- Multi-valued Attribute


- Composite Attributes

Key Attributes

Company Database Chapter Example

The company database keeps track of a company's employees, departments and projects. Suppose that after the requirements collection and analysis phase, the database designers provided the following description of the part of the company to be represented by the database.

1. The company is organized into department. Each department has a unique name a unique number and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
2. A department controls a number of projects, each of which has a unique name, a unique number and a single location.
3. We store each employees name, ID number, address, salary, sex and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date and relationship to the employee.

From the information given above, we can identify 4 entities.

1. **Department** – Name, Number, Locations, Manager, and Manager Start Date.
2. **Project** – Name, Number, Location and Controlling Department.
3. **Employee** – Name, ID Number, Sex, Address, Salary, Birth Date, Department. Both Name and Address can be a composite attribute, however it was not specified in the requirements.
4. **Dependent** – Employee, Dependent Name, Sex, Birth Date, Relationship

The information about the projects an employee works on can be represented in two ways. One, we can include a multi-valued composite, attribute, WorksOn(Project, Hours) in the Employee entity, or we can include a multi-valued composite attribute, Workers(Employee, Hours).

Relationship Types, Relationship Sets, Roles and Structural Constraints

- Looking at the example above, there are several implicit relationships among the entity types.
- Whenever an attribute of one entity type refers to another entity type, some relationship exists.
- For example, Manager of a department refers to an employee who manages the department, Controlling Department of the project, refers to the department that controls the project. Supervisor of an employee refers to the employee who supervises that employee.
- In an ER diagram, these references are not represented as attributes, but as relationships.

Relationship Types, Sets and Instances

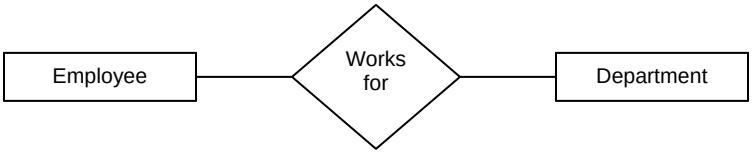
- A relationship type, R, among entities, defines a *relationship set* among entities from the entity types.

Role Names

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- The role name shows the role that an particular entity from the entity type plays in each relationship.
- Example: In the Company diagram, in the WorksFor relationship type, the employee plays the role of employee or worker, and the department entity plays the role of department or employer.
- In some cases the same entity participates more than once in a relationship type in different roles.
- For example, the Supervision relationship type relates an employee to a supervisor, where both the employee and supervisor are of the same employee entity type, therefore the employee entity participates twice in the relationship, once in the role of supervisor, and once in the role of supervisee.

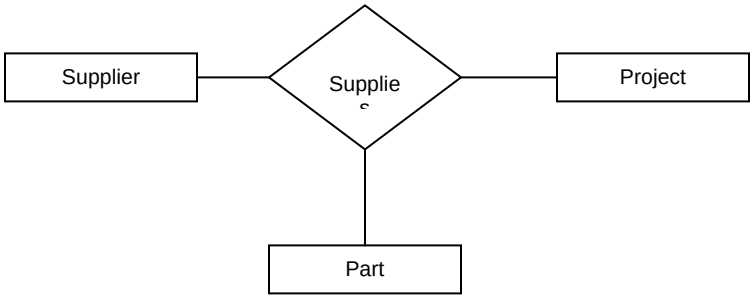
Relationship Review

- Each time an attribute of one entity type refers to another entity type, some relationship exists.
- In ER diagrams, these references should be represented as relationships, rather than attributes.
- For example, in the Company database schema, an attribute of employee is the department they work for, rather than representing this information as an attribute of the Employee entity type, it should be represented on a diagram as a relationship between the two entities.
- Relationships between entities are represented using a diamond shape.
- Relationships are usually given a verb name, which specifies the relationship between two entities.
- If we look at the relationship between Employee and Department, an employee works for a department, therefore the relationship would be represented



Degree of Relationship Type

- The degree of a relationship type is the number of participating entity types. Meaning if the relationship is between two entity types (Employee and Department), then the relationship is binary, or has a degree of two.
- If the relationship is between three participating entities, it has a degree of three, and therefore is a ternary relationship.
- For example, if we have three entities, Supplier, Project and Part. Each part is supplied by a unique supplier, and is used for a given project within a company; the relationship “Supplies” is a ternary (degree of three) between Supplier, Project and Part, meaning all three participate in the supplies relationship.

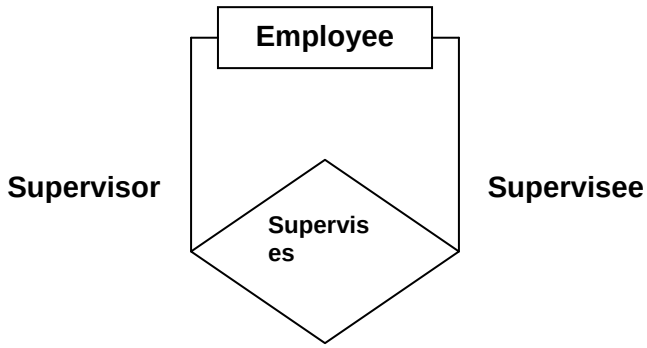


Role Names and Recursive Relationships

- Each entity type in a relationship plays a particular role. The role name specifies the role that a participating entity type plays in the relationship and explains what the relationship means.

- For example, in the relationship between Employee and Department, the Employee entity type plays the employee role, and the Department entity type plays the department or employer role.
- In most cases the role names do not have to be specified, but in cases where the same entity participates more than once in a relationship type in different roles.

For example, in the Company schema, each employee has a supervisor, we need to include the relationship "Supervises", however a supervisor is also an employee, therefore the employee entity type participates twice in the relationship, once as an employee and once as a supervisor, therefore we can specify two roles, employee and supervisor.



Constraints on Relationship Types

- Relationship types have certain constraints that limit the possible combination of entities that may participate in relationship.
- An example of a constraint is that if we have the entities Doctor and Patient, the organization may have a rule that a patient cannot be seen by more than one doctor. This constraint needs to be described in the schema.
- There are two main types of relationship constraints, cardinality ratio, and participation.

Cardinality for Binary Relationship

- Binary relationships are relationships between exactly two entities.
- The cardinality ratio specifies the maximum number of relationship instances that an entity can participate in.
- The possible cardinality ratios for binary relationship types are: 1:1, 1:N, N:1, M:N.
- Cardinality ratios are shown on ER diagrams by displaying 1, M and N on the diamonds.
- The ratio shown closest to an entity, represents the ratio the other entity has to that entity.

Participation Constraints and Existence Dependencies

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- The constraint specifies the minimum number of relationship instances that each entity can participate in.
- There are two types of participation constraints:
    - **Total:**

- If an entity can exist, only if it participates in at least one relationship instance, then that is called total participation, meaning that every entity in one set, must be related to at least one entity in a designated entity set.
- An example would be the Employee and Department relationship. If company policy states that every employee must work for a department, then an employee can exist only if it participates in at lest one relationship instance (i.e. an employee can't exist without a department)
- It is also sometimes called an existence dependency.
- Total participation is represented by a double line, going from the relationship to the dependent entity.

o **Partial**:
- If only a part of the set of entities participate in a relationship, then it is called partial participation.
- Using the Company example, every employee will not be a manager of a department, so the participation of an employee in the "Manages" relationship is partial.
- Partial participation is represented by a single line.

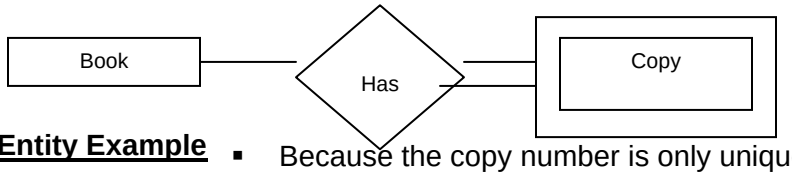
Attributes of Relationship Types

- Relationships can have attributes similar to entity types.
- For example, in the relationship Works_On, between the Employee entity and the Department entity we would like to keep track of the number of hours an employee works on a project. Therefore we can include Number of Hours as an attribute of the relationship.
- Another example is for the "manages" relationship between employee and department, we can add Start Date as an attribute of the Manages relationship.
- For some relationships (1:1, or 1:N), the attribute can be placed on one of the participating entity types. For example the "Manages" relationship is 1:1, StartDate can either be migrated to Employee or Department.

Weak Entity Types

- Entity types that do not have key attributes are called weak entity types.
- Entities that belong to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- This entity type is called an identifying or owner entity type.
- The relationship that relates the identifying entity type with the weak entity type is called an identifying relationship.
- A weak entity type always has a total participation constraint with respect to the identifying relationship, because a weak entity cannot exist without its owner.
- Not all existence dependencies result in a weak entity type; if an entity has a key attribute then it is not a weak entity.

- A weak entity type usually has a partial key, which is the set of attributes that can uniquely identify weak entities that are ***related to the same owner entity.***
- For example, lets assume in a library database, we have an entity type Book. For each book, we keep track of the author, ISBN, and title. The library may own several copies of the same book, and for each copy, it keeps track of the copy number (a different copy number for each copy of a given book) and price of each copy.

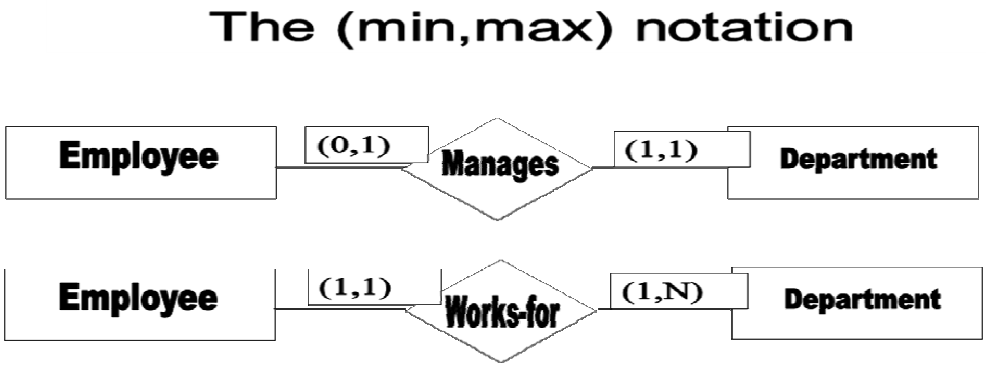


Weak Entity Example

- Because the copy number is only unique for each book (meaning Book 123 may have copy 1, copy 2, copy 3, and book 456 may also have copy 1, copy 2 and copy 3) and not for all copies of all books, it cannot be considered unique for each copy.
- Therefore because the Copy entity does not have a key attribute, it is considered a weak entity type, an is identified by being related to the Book entity. The book entity is the identifying entity, and the relationship is the identifying relationship.
- Because a copy cannot exist without the owner (Book) the Copy entity type has a total participation constraint with respect to the identifying relationship.
- The partial key of the Copy entity is Copy Number, for each owner entity Book, the Copy Number uniquely identifies the copy.

Min-Max Notation

- Before we saw that to specify structural constraints (cardinality) we used the M:N notation.
- An alternate notation involves specifying a pair of integers, which are used to specify the minimum and maximum participation of each entity type in the form of (min, max)
- A minimum participation of 0 indicates partial participation (meaning that there may be some entities that do not participate in the relationship)
- A minimum participation of 1 or more indicates total participation, meaning that each entity must participate in exactly/at least one-relationship type.
- See PowerPoint “ERD_Examples.ppt”, Slide 1 to demonstrate.



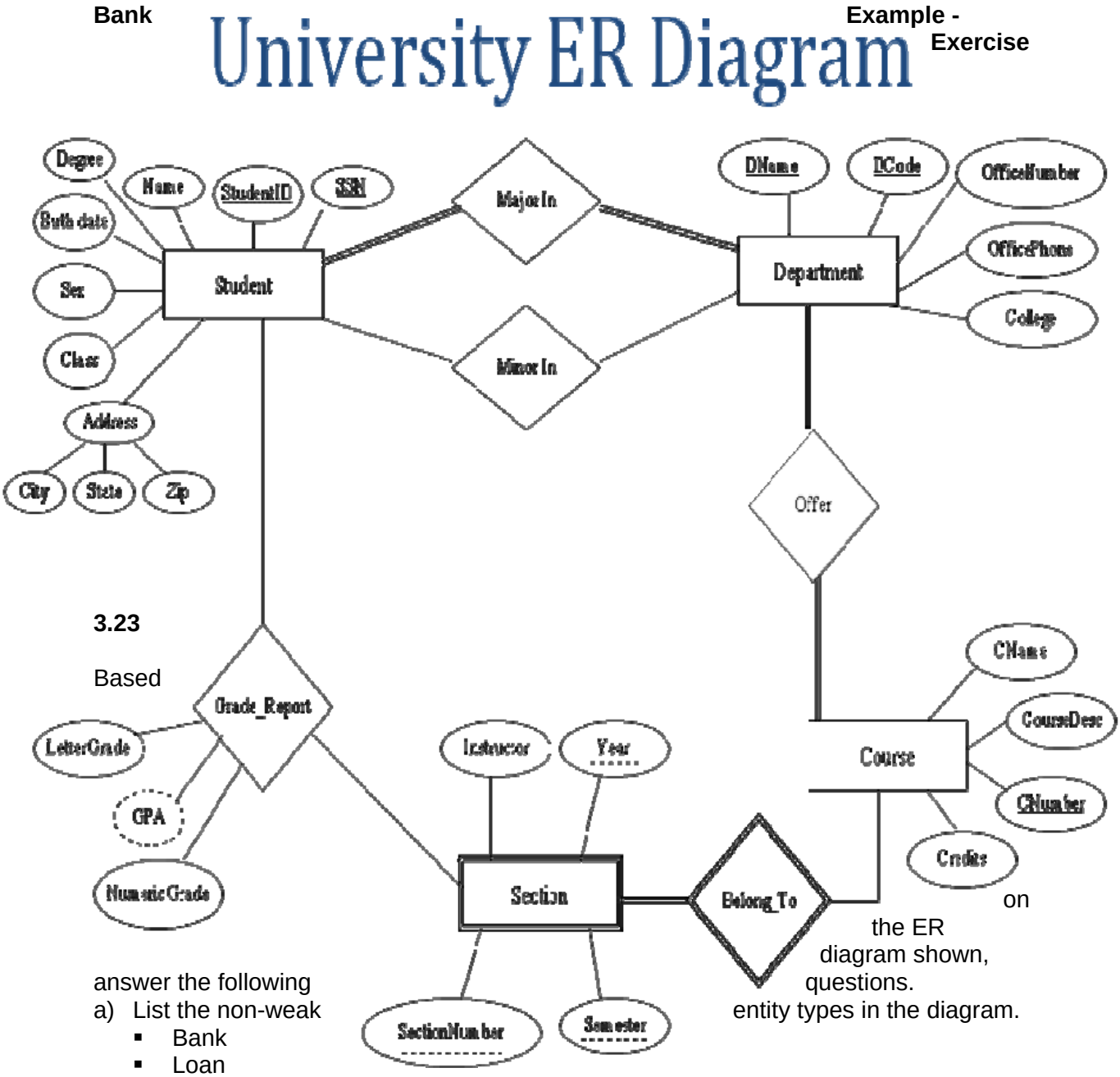
Examples

University Example – Exercise 3.16

Consider the following set of requirements for a university database that is used to keep track of student's transcripts:

- The university keeps track of each student's name, student number, social security number, current address and phone number, permanent address and phone number, birthdate, sex, class (freshman, graduate), major department, minor department (if any), degree program (B.A., B.S., ... Ph.D.). Some user applications need to refer to the city, state, and zip code of the student's permanent address and to the student's last name. Both social security number and student number are unique for each student. All students will have at least a major department.
- Each department is described by a name, department code, office number, office phone, and college. Both the name and code have unique values for each department.
- Each course has a course name, description, course number, number of credits, level and offering department. The course number is unique for each course.
- Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its value is an integer (1, 2, 3, ... up to the number of sections taught during each semester).
- A grade report must be generated for each student that lists the section, letter grade, and numeric grade (0,1,2,3, or 4) for each student and calculates his or her average GPA.

Bank **Example - Exercise**



3.23

Based on the ER diagram shown, answer the following questions.

a) List the non-weak entity types in the diagram.
   - Bank
   - Loan
   - Account
   - Customer

b) For any weak entities, list the name, partial key, and identifying relationship.
   - Name: Bank-Branch
   - Partial Key: Branch No
   - Identifying Relationship: Branches

c) What constraints do the partial key and the identifying relationship of the weak entity type specify in the diagram?

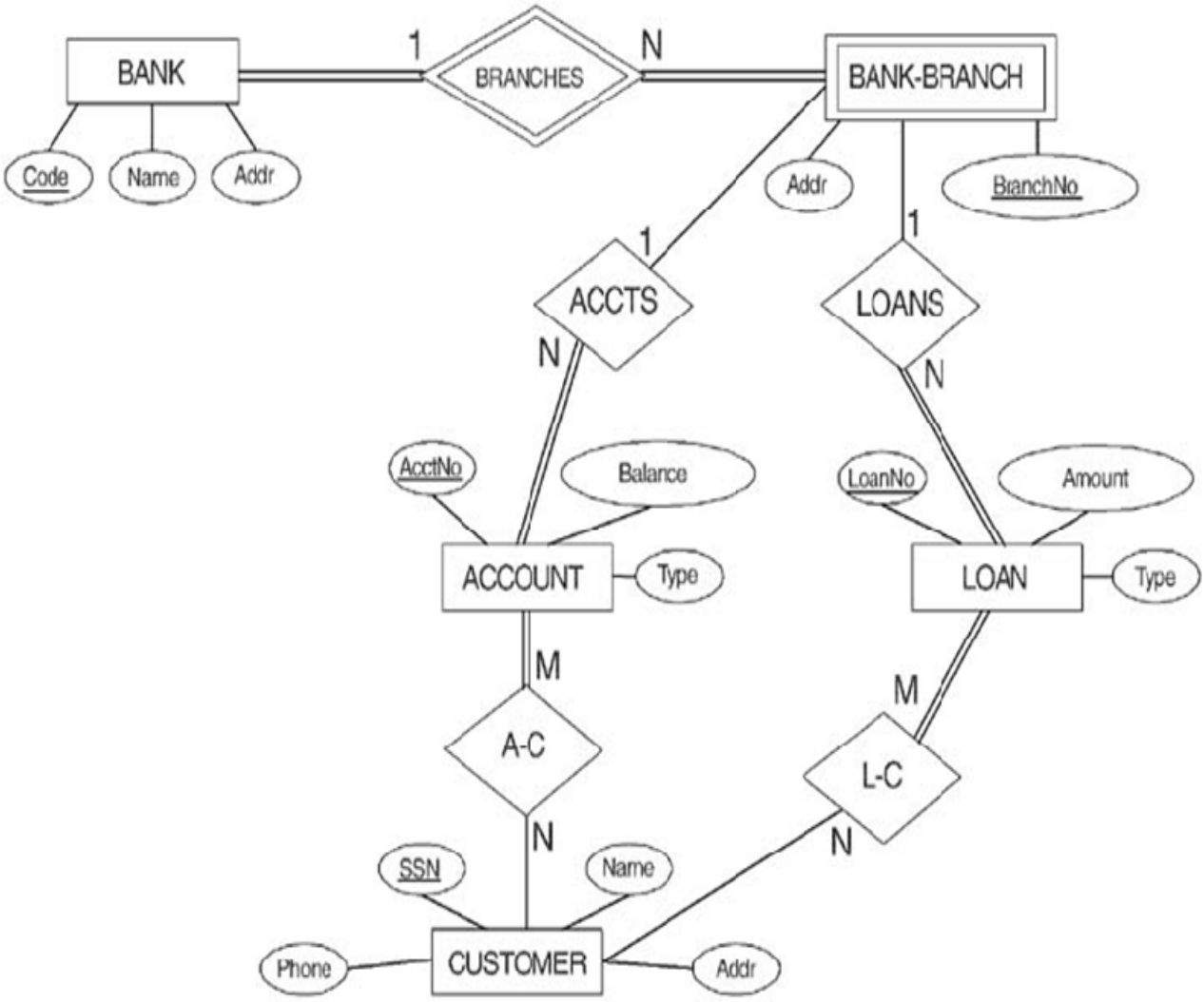
- Every bank must have at least one branch, and each bank branch belongs to one and only one bank. Together the branch number and bank code uniquely identify a branch.

d) List the names of all relationship types, and specify the (min, max) constraint on each participation of an entity type in a relationship type.
   - Branches, Accts, Loans, A-C, L-C.
   - See diagram for constraints.

e) List the user requirements that lead to the diagram.

f) Suppose that each customer must have at least one account, but is restricted to at most two loans at a time, and that a bank branch cannot have more than 1000 loans. How does this show up as min-max constraints.
   - Customer-Loan (0, 2)
   - Customer-Account (1, N)
   - Bank-Branch – Loan (0, 1000)

ER DIAGRAM FOR A BANK DATABASE

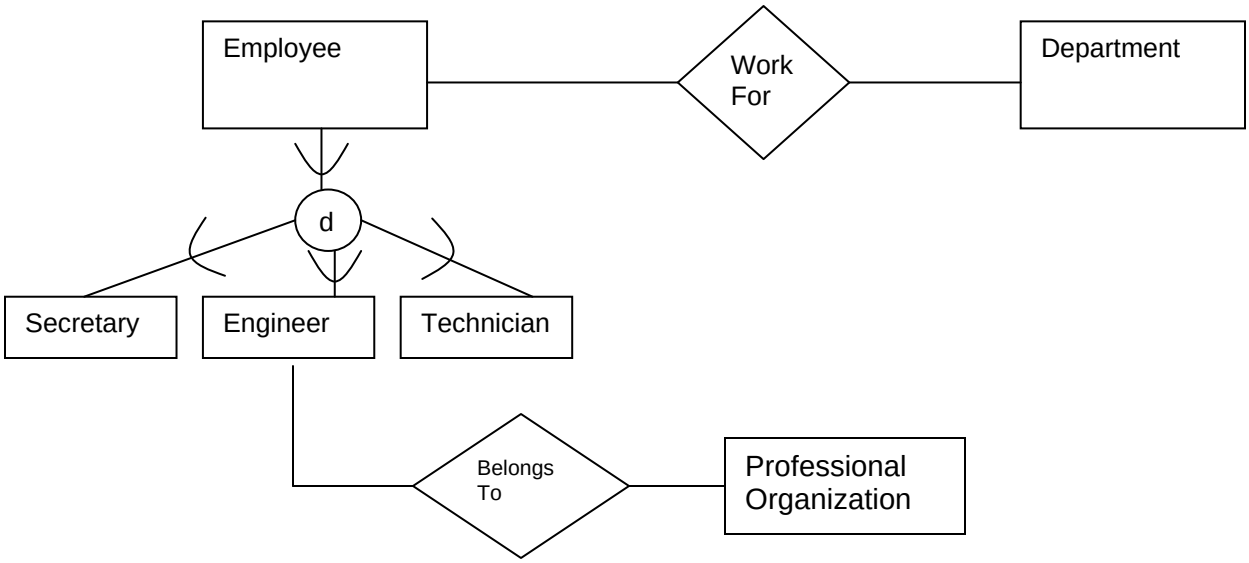


Specialization

- The process of defining a set of subclasses of a super class.
- Specialization is the top-down refinement into (super) classes and subclasses
- The set of sub classes is based on some distinguishing characteristic of the super class.
- For example, the set of sub classes for Employee, Secretary, Engineer, Technician, differentiates among employee based on job type.
- There may be several specializations of an entity type based on different distinguishing characteristics.
- Another example is the specialization, Salaried_Employee and Hourly_Employee, which distinguish employees based on their method of pay.

Notation for Specialization

- To represent a specialization, the subclasses that define a specialization are attached by lines to a circle that represents the specialization, and is connected to the super class.
- The subset symbol (half-circle) is shown on each line connecting a subclass to a super class, indicates the direction of the super class/subclass relationship.
- Attributes that only apply to the sub class are attached to the rectangle representing the subclass. They are called specific attributes.
- A sub class can also participate in specific relationship types. See Example.



Reasons for Specialization

- Certain attributes may apply to some but not all entities of a super class. A subclass is defined in order to group the entities to which the attributes apply.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass.

Summary of Specialization

Allows for:

- Defining set of subclasses of entity type
- Create additional specific attributes for each sub class
- Create additional specific relationship types between each sub class and other entity types or other subclasses.

Attribute-defined specialization

Generalization

- The reverse of specialization is generalization.
- Several classes with common features are generalized into a super class.
- For example, the entity types Car and Truck share common attributes License_PlateNo, VehicleID and Price, therefore they can be generalized into the super class Vehicle.

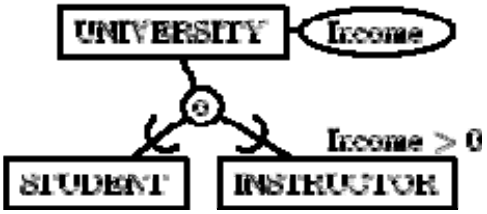
Constraints on Specialization and Generalization

- Several specializations can be defined on an entity type.
- Entities may belong to subclasses in each of the specializations.
- The specialization may also consist of a single subclass, such as the manager specialization, in this case we don't use the circle notation.

Types of Specializations

- Occurs in cases where we can determine exactly the entities of each sub class by placing a condition of the value of an attribute in the super class.
- An example is where the Employee entity has an attribute, Job Type. We can specify the condition of membership in the Secretary subclass by the condition, JobType="Secretary"

Another Example:



- The condition is called the defining predicate of the sub class.
- The condition is a constraint specifying exactly those entities of the Employee entity type whose attribute value for Job Type is Secretary belong to the subclass.
- Predicate defined subclasses are displayed by writing the predicate condition next to the line that connects the subclass to the specialization circle.

- If all subclasses in a specialization have their membership condition on the same attribute of the super class, the specialization is called an attribute-defined specialization, and the attribute is called the defining attribute.
- Attribute-defined specializations are displayed by placing the defining attribute name next to the arc from the circle to the super class.

Attribute-defined specialization

- If all subclasses in a specialization have their membership condition on the same attribute of the super class, the specialization is called an attribute-defined specialization, and the attribute is called the defining attribute.
- Attribute-defined specializations are displayed by placing the defining attribute name next to the arc from the circle to the super class.

User-defined specialization

- When we do not have a condition for determining membership in a subclass the subclass is called user-defined.
- Membership to a subclass is determined by the database users when they add an entity to the subclass.

Disjointness/Overlap Constraint

- Specifies that the subclass of the specialization must be disjoint, which means that an entity can be a member of, at most, one subclass of the specialization.
- The d in the specialization circle stands for disjoint.
- If the subclasses are not constrained to be disjoint, they overlap.
- Overlap means that an entity can be a member of more than one subclass of the specialization.
- Overlap constraint is shown by placing an o in the specialization circle.

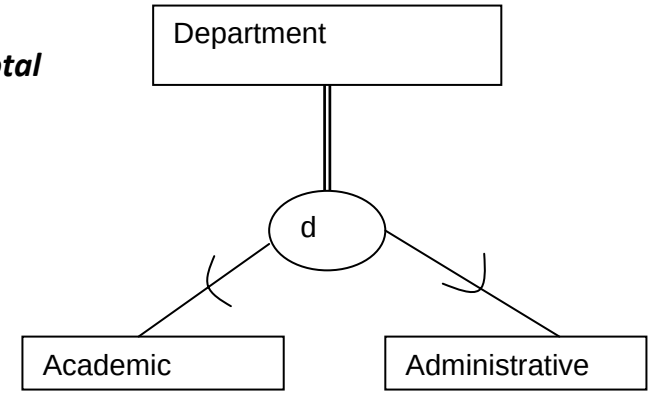
Completeness Constraint

- The completeness constraint may be either total or partial.
- A **total specialization** constraint specifies that every entity in the superclass must be a member of at least one subclass of the specialization.
- Total specialization is shown by using a double line to connect the super class to the circle.
- A single line is used to display a **partial specialization**, meaning that an entity does not have to belong to any of the subclasses.

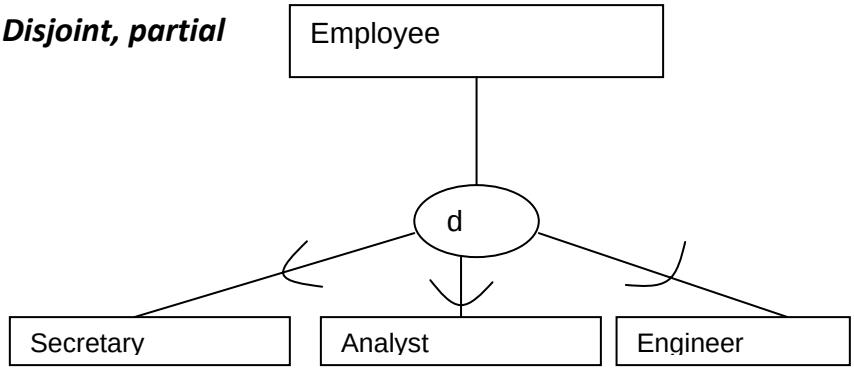
Disjointness vs. Completeness

- Disjoint constraints and completeness constraints are independent. The following possible constraints on specializations are possible:

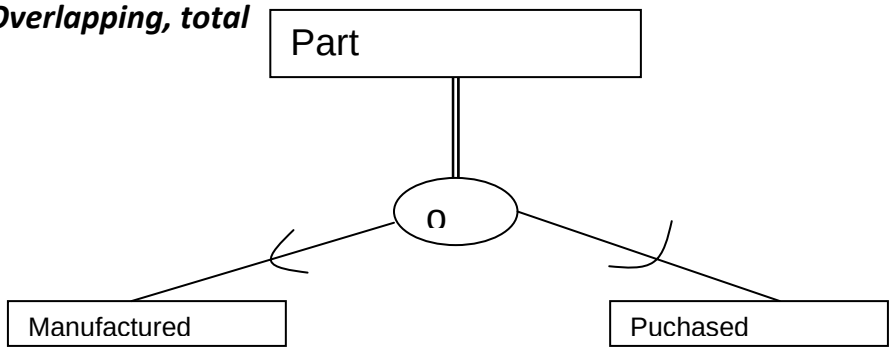
Disjoint, total



Disjoint, partial



Overlapping, total



Overlapping, partial