Conceptual Design: The Entity-Relationship Model

Databases

2018-2019

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

.

Bibliography

- Basic Bibliography:
  - R. Elmasri, S.B. Navathe. **Fundamentals of Database Systems** (6th ed). Addison-Wesley, 2010. (in Spanish: **Fundamentos de Sistemas de Bases de Datos** (5a ed). Addison-Wesley, 2007).
    Chapters 3 and 4 (5th ed) or 7 and 8 (6th ed).
- Additional Bibliography:
  - A. Silberschatz , H. F. Korth, S. Sudarshan. **Database Systems Concepts** (5th ed) McGraw-Hill, 2006. (in Spanish: **Fundamentos de Bases de Datos** (5a ed) McGraw-Hill, 2006).
    Chapter 6. (in particular the EER model: Section 6.7).

Contents

- Introduction: The entity-relationship model.
- Entities and attributes.
  - Superkeys and candidate keys. Weak entities.
  - Entity type.
- ER diagrams.
- Relations.
  - Relationship attributes.
  - Relationship type.
  - Degree of a relationship.
  - Recursive relationships.
  - Constraints: cardinality and participation.
  - Constraints and weak entities.
- Extended ER diagrams.
  - Specialization and generalization. Attribute inheritance.
  - Aggregations.

Introduction. The Entity-Relationship Model

- We can use this model to represent the information in a DB at the **conceptual level.**
- It is obtained from the initial analysis of the system, **although this process is not straightforward.**
  - A good conceptual design requires the study of the system requirements and the analysis information.
- The design of a DB must be **concise**, **easy to understand and maintain**, and should allow an **efficient** logical and physical design.
- During the system analysis several *candidate* design models can be obtained for the same system.
- The fundamental elements of the ER model are:
  - **Entities,** entity types.
  - **Attributes.**
  - **Relationships,** relationship types and **constraints** on relationships.
- ER models are graphically represented by **ER diagrams.**

Introduction. The Entity-Relationship Model

- Steps for designing the ER model:
  - Select entity types and their attributes.
  - Select relationship types.
  - Define constraints.
- We introduce the elements of the ER model with an **example:**
  - We are in charge of designing a DB for managing a food delivery services company.

Introduction. Initial example

- The specialty of our company:

- Some of the requirements for the system follow:

Introduction. Initial Example Requirements

- We have to keep some data of the **employees** working at this company: name, SS number, position (cook, kitchen helper, waiter, rider...), phone number (landline, cellphone), age.
- The company has several **branches**.
- there exists a hierarchical relationship between employees.
- The company produces several **dishes**, each one identified by a code, with a description (maki roll set, nigiri&sashimi set, etc.) and its price.
- For preparing each dish we need to know the **ingredients** required and their amounts for cooking the dish.
- All employees must be able to prepare at least one dish and at most 8 dishes.
- We need to know which dishes can be prepared by each employee.
- We have to keep information about **customers** (phone number, address, email, orders requested).
- Each **order** is identified by a number and we have to keep some information about the order: date, requested dishes, quantity.
- We have to know the **supplier** that provides each of the ingredients in each branch, etc, etc.

Entities

- Entities are the basic elements of the ER model.
- They represent **one "object" of the real world**, distinguishable from all other "objects" in the DB.
- It can be either a physical "object" (a vehicle, a person) or a concept (a course, a financial loan, an order).
- A particular entity is defined by a **set of attributes:** properties that describe it.
- An **entity type** defines a **set of entities that are defined by the same attributes.**
- Every entity has a specific value for each of its attributes.
- **Example:** An entity "employee" with the following attributes:

SSN:	1234567890
Name:	Andrés Sánchez García
Position:	Kitchen helper
Age:	25
Cellphone number:	600123456

Attributes

- An attribute has a **domain** associated to it: a set of valid values that the attribute can take in any entity.
- Each entity in an entity type has a value associated to each attribute.
- There are several kinds of attributes:
  - **Simple** (atomic) or **composite** attributes. Composite attributes are composed of smaller attributes (e.g., personal name, composed of given name and family name).
  - **single-valued** or **multi-valued** attributes. Multi-valued attributes can take several values for a particular entity (e.g., the color of a car might be a multi-valued attribute).
  - **stored** or **derived** attributes. Derived attributes can be computed from other attributes or entities (e.g., age can be computed from the birth date and system current date).
- If an entity has no value for a particular attribute, it takes the **null** value. For example, if an employee has no telephone.

Superkeys, candidate keys, primary key

- Entities of an entity type must be **distinguishable** from the rest of entities of that type.
  - As entities are defined by the values of their attributes, some attributes must be used to identify any particular entity in an entity type.
- A **superkey** is a subset of the attributes of the entity type that **uniquely identify every entity in the entity type.**
  - Any superset of a superkey is also a superkey.
  - But we are interested in a minimum-size set of attributes that identifies each entity.
- A **candidate key** is a superkey that does not contain any subset that is also a superkey.
- There can exist several candidate keys for an entity type. The **primary key** of the entity type is one of them, chosen by the designer of the DB.

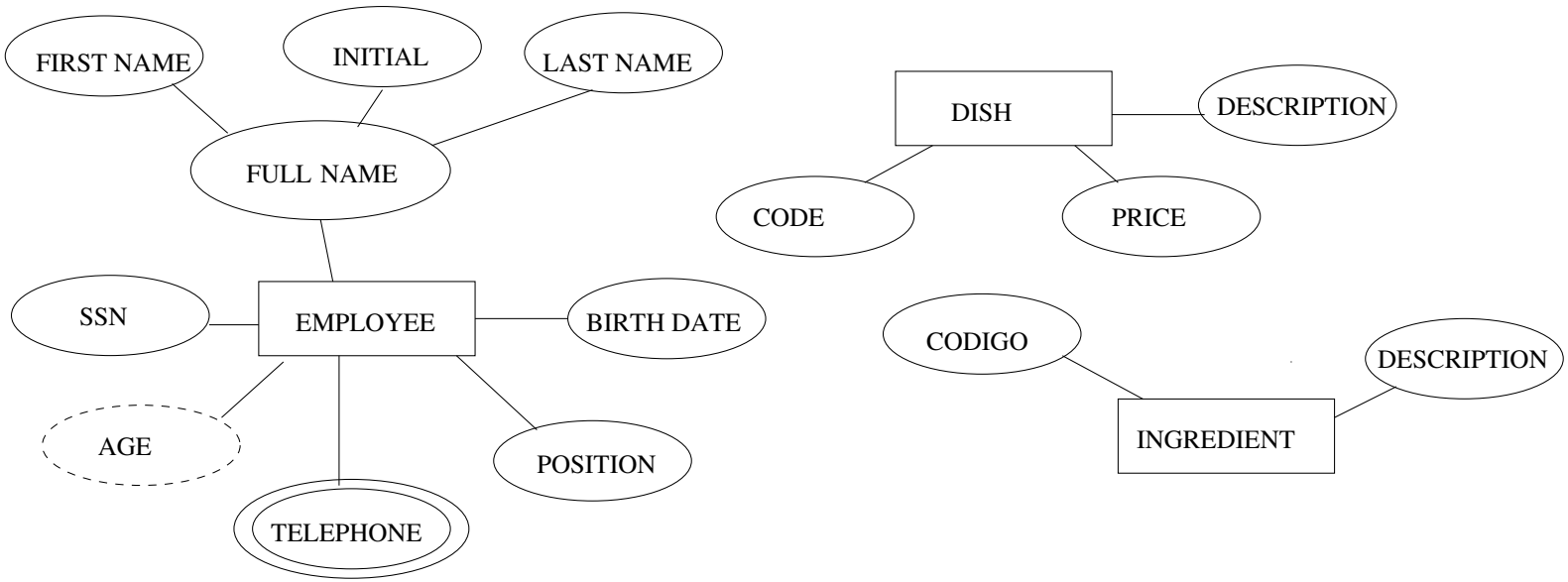
Superkeys, candidate keys, primary key

- **Example:** Entity type `employee`:
  - Superkeys: {`NIF`}, {`SSN`}, {`cellphone`}, {`NIF, Name`}, {`SSN, Name, Age`},...
  - {`Age`}, {`Name, Age`}... are **not** superkeys.
  - Candidate keys: {`NIF`}, {`SSN`}, {`cellphone`}
- Each entity type usually has **one attribute which is a candidate key on its own**, but it may be the case that a primary key is composed of more than one attribute.
- An entity type is **weak** if there is no set of its attributes that is a superkey. We will study this case later.

ER Diagrams

- An **ER Diagram** is a graphical representation of the structure of a DB at a conceptual level.
- Basic elements:
  - **Entity type:** is represented by a box (rectangle).
    - ⋆ **Weak** entities are represented by **double rectangles.**
  - **attribute:** is represented by an ellipse connected to its entity type (or relationship) by a line.
    - ⋆ Attributes in the **primary key** are **underlined.**
    - ⋆ **Multi-valued** attributes use a **double ellipse.**
    - ⋆ Each component of a composite attribute is represented as an attribute connected to the composite attribute by a line.
    - ⋆ **Derived** attributed are represented by **dashed ellipse.**
  - **Relationship type:** is represented by a **rhombus** connected to the related entities by lines.

ER Diagram example: entities and attributes



EMPLOYEE:
$\{\langle 0101, (John, P, Doe), 21, \{666111222, 911234567\}, Kitchen\ Helper, 01.01.95\rangle,$
$\langle 0202, (Joseph, M, Smith), 26, \{666123231\}, Waiter, 01/01/90\rangle, ...\}$

DISH: $\{\langle PL001, California\ maki, 8.50€\rangle, \langle PL002, Nigiri, 3.00€\rangle, ...\}$

INGREDIENT: $\{\langle IN001, Salmon\rangle, \langle IN002, Rice\rangle, \langle IN003, Wasabi\rangle, ...\}$

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?

Relationships

- Entities in the example include *implicit relationships*. For example:
  between a dish and its ingredients.
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?**

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€,` **`IN001, salmon`** ⟩
  ⟨`PL001, California maki, 8.50€,` **`IN002, Rice`** ⟩

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€,` **`IN001, salmon`** ⟩
  ⟨`PL001, California maki, 8.50€,` **`IN002, Rice`** ⟩ → redundancy.

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€,` **`IN001, salmon`** ⟩
  ⟨`PL001, California maki, 8.50€,` **`IN002, Rice`** ⟩ → redundancy.
  (2) Make `ingredient` be a **multi-valued attribute:**
  ⟨`PL001, California maki, 8.50€,` {⟨**`IN001,Salmon`**⟩,⟨**`IN002,Rice`**⟩} ⟩
  ⟨`PL002, Nigiri, 3.00€,` {⟨**`IN003,Wasabi`**⟩,⟨**`IN002,Rice`**⟩,...} ⟩

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€, IN001, salmon` ⟩
  ⟨`PL001, California maki, 8.50€, IN002, Rice` ⟩ → redundancy.
  (2) Make `ingredient` be a **multi-valued attribute:**
  ⟨`PL001, California maki, 8.50€,` {⟨`IN001,Salmon`⟩,⟨`IN002,Rice`⟩} ⟩
  ⟨`PL002, Nigiri, 3.00€,` {⟨`IN003,Wasabi`⟩,⟨`IN002,Rice`⟩,`...`} ⟩
  - ★ Each ingredient has attributes → redundancy.

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€,` **`IN001, salmon`** ⟩
  ⟨`PL001, California maki, 8.50€,` **`IN002, Rice`** ⟩ → redundancy.
  (2) Make `ingredient` be a **multi-valued attribute:**
  ⟨`PL001, California maki, 8.50€,` {⟨**`IN001,Salmon`**⟩,⟨**`IN002,Rice`**⟩} ⟩
  ⟨`PL002, Nigiri, 3.00€,` {⟨**`IN003,Wasabi`**⟩,⟨**`IN002,Rice`**⟩,...} ⟩
  - ★ Each ingredient has attributes → redundancy.
  - ★ The same ingredient may be in several dishes → redundancy.

Relationships

- Entities in the example include *implicit relationships*. For example: **between a dish and its ingredients.**
  How can we solve this using basic elements of ER diagrams?
- **Making `ingredient` an attribute of `dish`?** A dish has **several ingredients**. Possible solutions:
  (1) Repeat the dish for each ingredient:
  ⟨`PL001, California maki, 8.50€, IN001, salmon` ⟩
  ⟨`PL001, California maki, 8.50€, IN002, Rice` ⟩ → redundancy.
  (2) Make `ingredient` be a **multi-valued attribute:**
  ⟨`PL001, California maki, 8.50€, {⟨IN001,Salmon⟩,⟨IN002,Rice⟩}` ⟩
  ⟨`PL002, Nigiri, 3.00€, {⟨IN003,Wasabi⟩,⟨IN002,Rice⟩,...}` ⟩
  - ★ Each ingredient has attributes → redundancy.
  - ★ The same ingredient may be in several dishes → redundancy.
  - Furthermore, an ingredient can only be **stored in the DB** if it is related to a dish.
- **Solution:**

Refine the design and **establish relationships between entities.**

Relationships

A **relationship** is an **association between entities.**

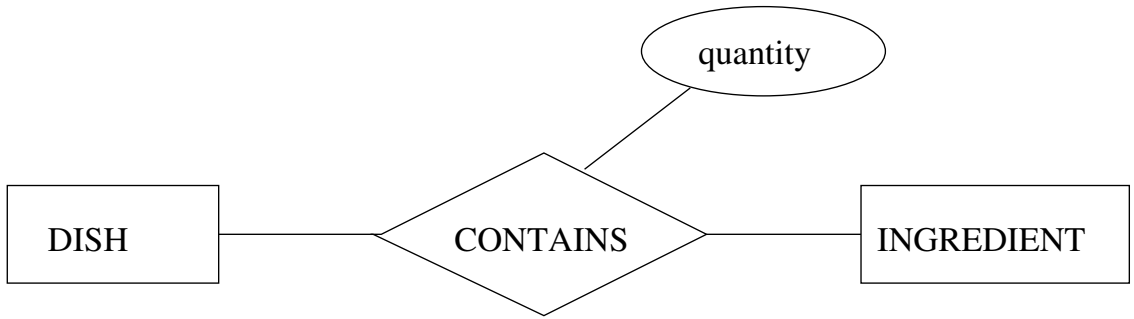
- A **relationship type** between entity types $E_1, \ldots, E_n$ define the **associations between entities of each entity type.**
- It is a **mathematical relationship**: it is **a subset** of the cartesian product $E_1 \times \cdots \times E_n$:

$$\{(e_1, \ldots, e_n) \mid e_1 \in E_1, \ldots, e_n \in E_n\}$$

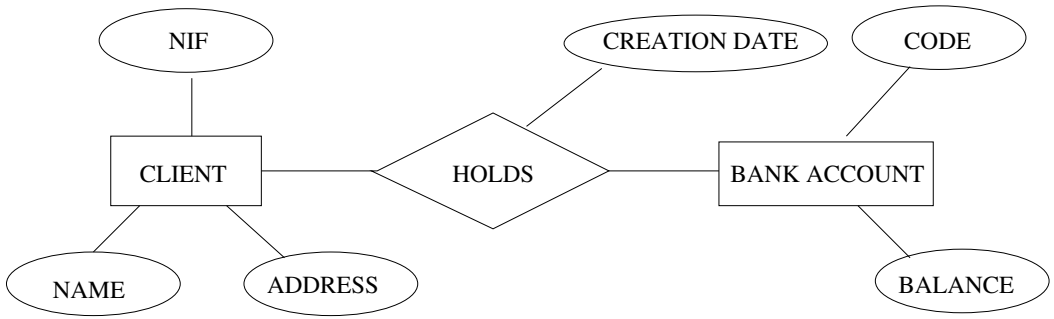
- Each element of this set is an **instance of the relationship type.**
- **Example:** we can define a relationship ``contains'' between the entity types `dish` and `ingredient`:

Relationship attributes

- A relationship **may have attributes:**
- Each dish may contain a certain ingredient in a quantity different from other dishes: `quantity` is a **relationship attribute.**

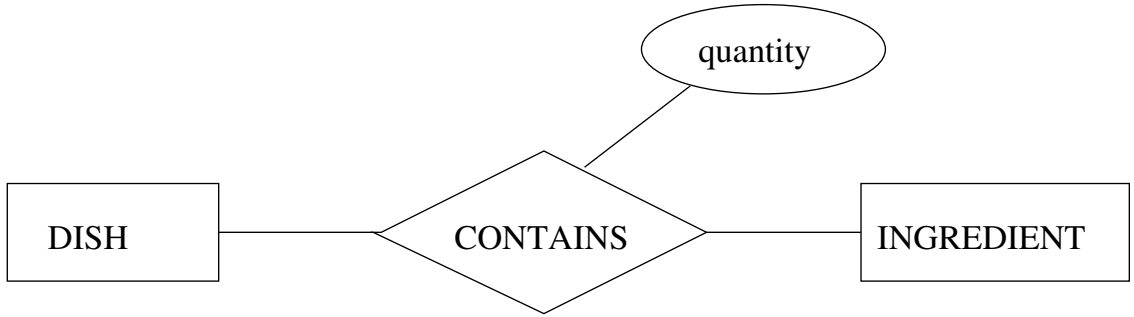


- Attributes should only be associated to a relationship if they do not fit in any of the participant entities:

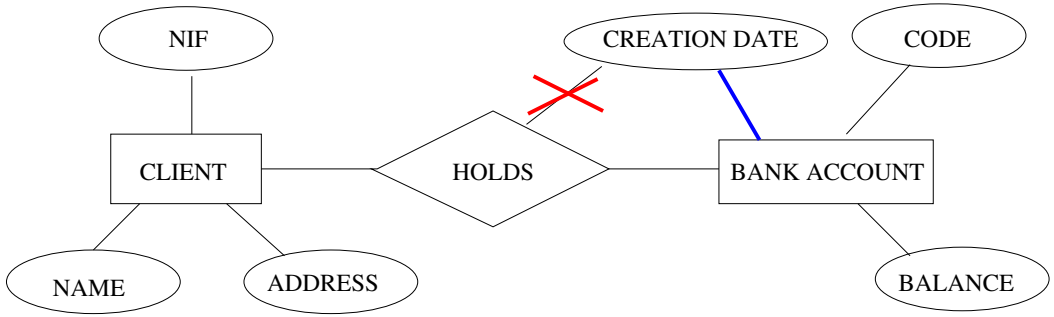


Relationship attributes

- A relationship **may have attributes:**
- Each dish may contain a certain ingredient in a quantity different from other dishes: `quantity` is a **relationship attribute.**



- Attributes should only be associated to a relationship if they do not fit in any of the participant entities:



Binary and ternary relationships

- Relationships can connect **two or more entities.** Let us see an **example:**
- We want the DB to store information regarding purchases of ingredients to suppliers for each branch.
- Three entity types participate:
  - `ingredient`,
  - `branch`, with attributes `code` and `address`,
  - `supplier`, with attributes `Id`, `Name` and `address`.
- We have to add relationships between these entities to store the suppliers that supply each ingredient to each branch.
- We can create **two relationships:**
  - **`supplies`**, between `supplier` and `branch`,
  - **`provides`**, between `supplier` and `ingredient`.

Binary and ternary relationships

- However, with relationships **supplies** and **provides** **we are unable to answer** the following question: **Which supplier provides a given ingredient to a given branch?**
  - Each relationship contains part of the information on its own, but that information **cannot be merged together.**

ingredient:

{⟨IN001, Salmon⟩, ⟨IN002, Rice⟩, ⟨IN003, Wasabi⟩, ...}

branch:

{⟨BR01, (1450, Washington St, CA94109, San Francisco, USA)⟩,
⟨BR02, (2400, Central Ave SE, NM87106, Albuquerque, USA)⟩,...}

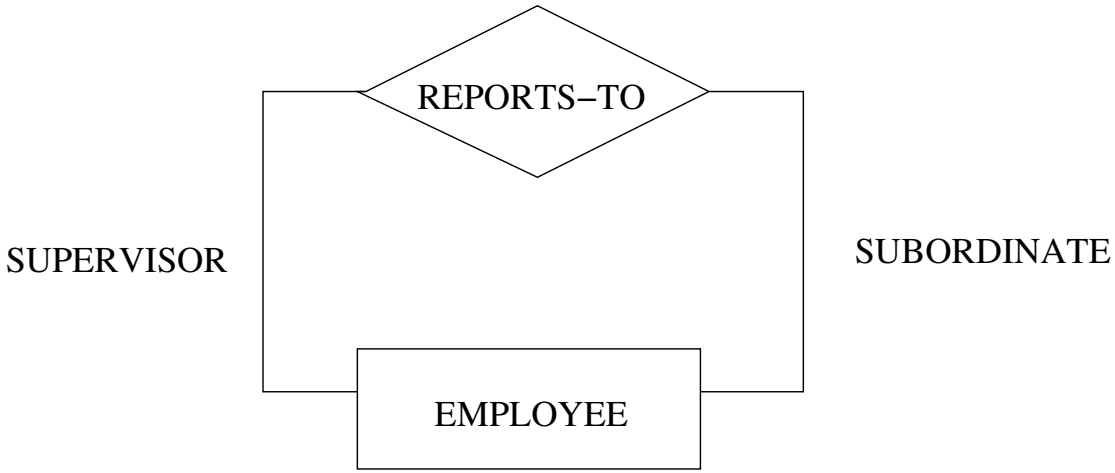
supplier:

{⟨PR01, AgroFarm, (1100, Laurel St, NM 87506, Santa Fe, USA)⟩,
⟨PR02, Sea Paradise, (5, Tsukiji, 1040045, Tokyo, Japan)...⟩}

- To solve this problem we have to set a **relationship between all three entities.**
- The **degree** of a relationship type is the number of participating entity types.

Recursive relationships

- A **recursive relationship** is a relationship type in which an entity type participates in the relationship more than once, acting under different **roles.**
- In this case we have to add a label to each line that connect the entity type with the relationship type to clarify the role of that line.
- Example: the relationship `reports-to` between an employee (the subordinate) and another employee (the supervisor).

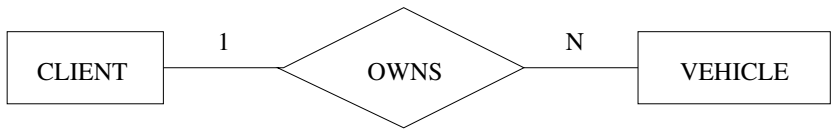


Constraints: cardinality and participation

- We can add information to ER Diagrams to represent **constraints** on the combinations of entities in the relationship set.
- They are used to limit the number of times that entities can participate in a relationship set.
- **Examples:**
  - "A dish contains ingredients (at least one)."
  - "Every employee must be able to prepare between 1 and 8 dishes."
  - (in a garage) "A client may own several vehicles, but a vehicle can have only one owner.".
- There exist two main types of constraints:
  - **cardinality constraints** (or *"cardinality ratio"*).
  - **Participation constraints.**

Cardinality Constraints

- Specify the **maximum number of relationships** in which any entity (in an entity type) can appear in the relationship set.
  Example: *"A client may own several vehicles, but a vehicle can have only one owner."*
  - An entity `client` may appear in relationships with several entities `vehicle`.
  - But an entity `vehicle` can appear in just one element of the relationship `owns` with an entity `client`.
- We can refer to particular or generic values for cardinality constraints:
  - There are many relationships in which we use cardinality **1**.
  - When we refer to *"several"* we use a generic **N**, where the exact value of N is not specified.
  - A particular case of "several" is a particular value greater than 1.



  - Notice **de location of the cardinality constraints** in this diagram.

Cardinality Constraints. Cardinality ratio

- The **cardinality ratio** for a binary relationship specifies the **maximum number of relationship instances** that an entity can participate in.
- **Example:** relationship `client:vehicle` is of cardinality ratio **1:N**: *each client owns any number of vehicles, but a vehicle can be owned by only one client*
- Possible cardinality ratios for a binary relationship between entities A and B (A:B) are:
  - **One to One (1:1):** each element in A is related to **at most one** element in B and vice versa.
  - **One to many (1:N):** each element in A is related to **any number** of elements in B, but each element in B is related to **at most one** element in A.
  - **Many to one (N:1):** each element in A is related to **at most one** element in B, but each element in B is related to **any number** of elements in A.
  - **Many to many (M:N):** each element in A is related to **any number** of elements in B and vice versa.

Cardinality constraints. Examples

- **One to one:** Relationship `manages` between `employee` and `department`: a department can be managed by only one employee (the head of the department), and an employee can be the head of at most one department.

- **One to many:** relationship `requests` between `customer` and `order`: a customer may request several orders, but an order can be requested by only one customer.

- **Many to many:** relationship `contains` between `dish` and `ingredient`: a dish may contain any number of ingredients, and an ingredient may be used for any number of dishes.

Cardinality Constraints. General rule

Let R be a binary relationship type between entity types E_1 and E_2. **The cardinality of E_1 is n if:**
given an entity $e_2 \in E_2$, there exist at most n entities $e_{11}, \ldots, e_{1n} \in E_1$ such that $\langle e_{1i}, e_2 \rangle \in R, 1 \leq i \leq n$.

- In this rule, n can be either 1 or N (any number), or even a particular number greater than 1.
- **Example:** "Every employee must be able to prepare between 1 and 8 dishes."[1]

[1] participation constraints are not shown in this diagram.

Cardinality constraints. Alternative notation

- **One to one:** Relationship `manages` between `employee` and `department`: a department can be managed by only one employee (the head of the department), and an employee can be the head of at most one department).

- **One to many:** relationship `requests` between `customer` and `order`: a customer may request several orders, but an order can be requested by only one customer.

- **Many to many:** relationship `contains` between `dish` and `ingredient`: a dish may contain any number of ingredients, and an ingredient may be used for any number of dishes.

Cardinality constraints in ternary relationships

- A ternary relationship may have many different cardinality ratios: M:N:Q, M:N:1 (and their permutations)[2].
- **Example:** In a ternary relationship between `branch`, `ingredient` and `supplier`:
  - If *"every supplier can provide any number of ingredients to any branch, and an ingredient can be provided by several suppliers to a branch"*:

    it is an M:N:Q relationship
  - But if *"a branch can be provided of an ingredient by only one supplier:*

    it is a M:N:1 relationship

    In this case, the "1" corresponds to the supplier.

Let R be a k-ary relationship. **The cardinality of E_i is n if:** given $e_1 \in E_1, \ldots, e_{i-1} \in E_{i-1}, e_{i+1} \in E_{i+1}, \ldots, e_k \in E_k$, there exist at most n entities $e_{i1}, \ldots, e_{in} \in E_i$ such that $\langle e_1, \ldots, e_{i-1}, e_{ij}, e_{i+1}, ..., e_k \rangle \in R, 1 \leq j \leq n$.

[2] We consider one entity with card. 1 at most (See Silberchatz, Sec. 6.4.)

Participation constraints

- A **participation constraint** denotes if every element in an entity type **must** participate in a relationship, or the participation is optional.
- Corresponds to the **minimum number** of instances of the relationship in which an entity must participate. There are two types:
  - **Total participation (1):** every entity in the entity type must participate in **at least one instance of the relationship set.**
    Example: *"Every order is requested by a customer."*
  - **Partial participation (0):** there may be some entities in the entity type that **do not appear in any instance of the relationship set.**
    Example: *"There might be ingredients that are not contained in any dish."*
- These constraints are represented in ER diagrams by means of a **double line** if the participation is **total,** or a **single line** if it is **partial.**

Participation constraints. Examples

- Between `customer` and `order`: *"a customer may request any number of orders (or even none), and an order is mandatorily associated to a single customer"*:

- Between `dish` and `ingredient`: *"Every dish contains at leas one ingredient, but there may be ingredients not contained in any dish"*:

Alternative notation combining cardinality and participation constraints

- Also called **(min,max) notation** (see Elmasri 6th ed, sec. 7.7.4).
- Combines participation and cardinality in a pair, **but changes the location of cardinality constraints.**
- **Examples:**
  - Between `customer` and `order`:

  - Between `dish` and `ingredient`:

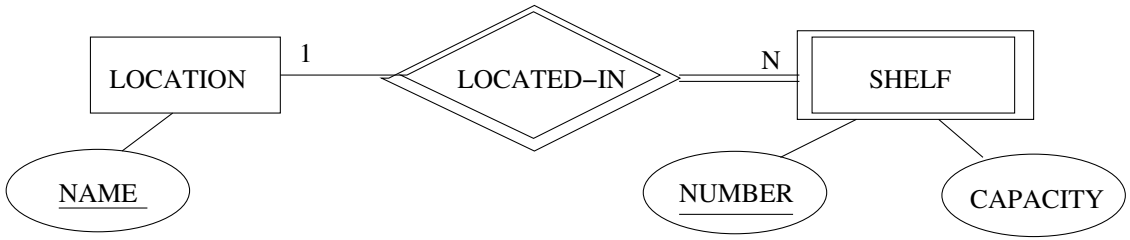
- It is **more expressive:** allows participation constraints different from 0 (partial) and 1 (total).
- It has **a different meaning** in ternary relationships.

Weak entity types

- A **weak entity type** is an entity type in which there is no set of attributes that can act as a primary key:
  - There can be several entities with exactly the same values for all attributes.
- Those entity types must be always associated to another entity type, named **identifying or owner entity type**, by means of an **identifying relationship.**
- The identifying relationship must always have the following characteristics:
  - **1:N** cardinality ratio (`owner:weak`), and
  - **total participation** constraint of the weak entity type.
- The weak entity type must have some attributes that form a **partial key** that distinguish the weak entities that correspond to each identifying entity.

Weak entity types

- Weak entity types are represented in the ER diagram using a **double box** and the identifying relationship is represented using a **double rhombus.**
- **Example:** *"Ingredients are stored in* **shelves** *that are located in several* **locations** *(refrigerator, pantry, ...). Shelves in a given location are numbered starting from 1. Therefore, there may be several shelves with the same number in different locations."*



Enhanced ER model

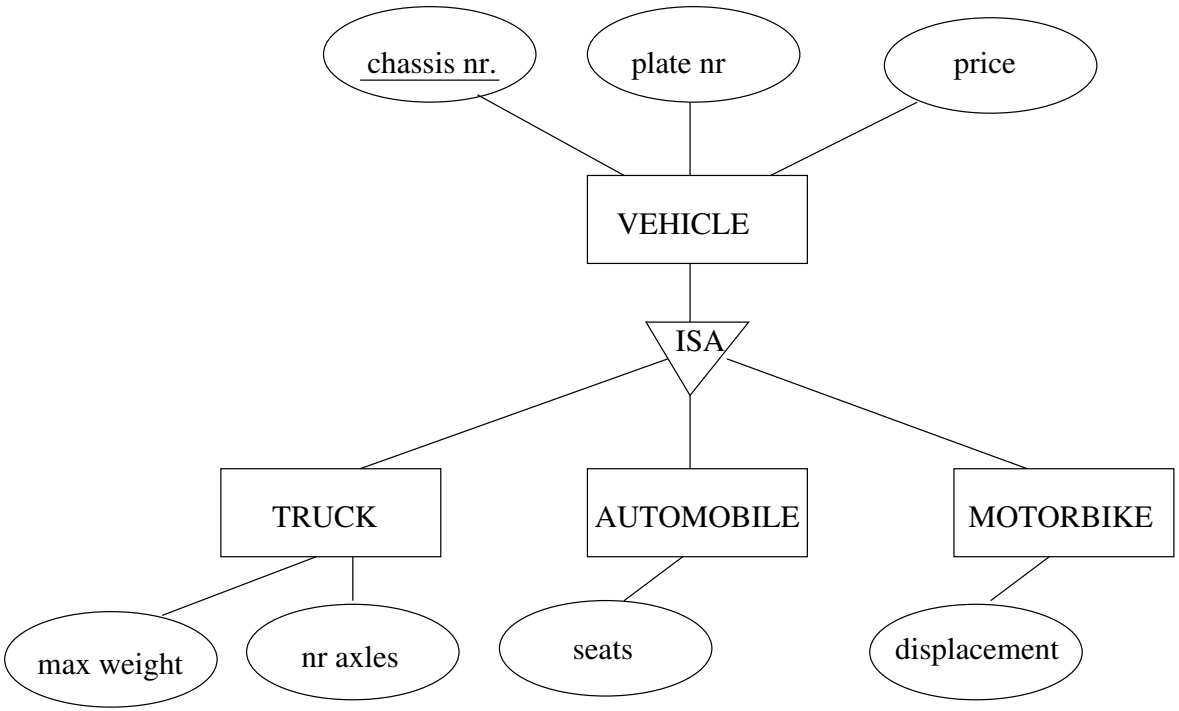
- The **enhanced ER model (EER)** adds additional elements to the ER model to provide more expressiveness to the design diagrams of a DB:
  - Specialization and generalization
  - Subclasses and superclasses
  - Attribute inheritance
  - Agregations

EER model. Specialization and generalization

- During the design of the ER model we can sometimes identify groups of entities in an entity type that share common characteristics, but that difer from other entities in the same type.
- **Specialization** is a **top-down** design technique:
  - Starts from an initial set of entities.
  - Identifies subgroups (**subclasses**) of entities that are **specializations** of the initial entity type.
- **Generalization** is the opposite, **bottom-up**, technique:
  - Starts from entity types that share common characteristics.
  - Identifies entity types that can be **generalized** in a single **superclass.**
- EER diagrams represent both techniques using the same element:
- an inverted triangle named **ISA** (*is - a*) to identify superclass / subclass relationships.
- Subclass entity types **inherit the attributes and relationships of the superclasses.**

EER model. Specialization and generalization. Example

- **Example:** We start a DB design using first the entity type `vehicle` and during the design phase we can specialize it for each type: `truck`, `automobile` or `motorbike`.
- Attributes and relationships of the higher level in the hierarchy **are inherited** by lower levels.



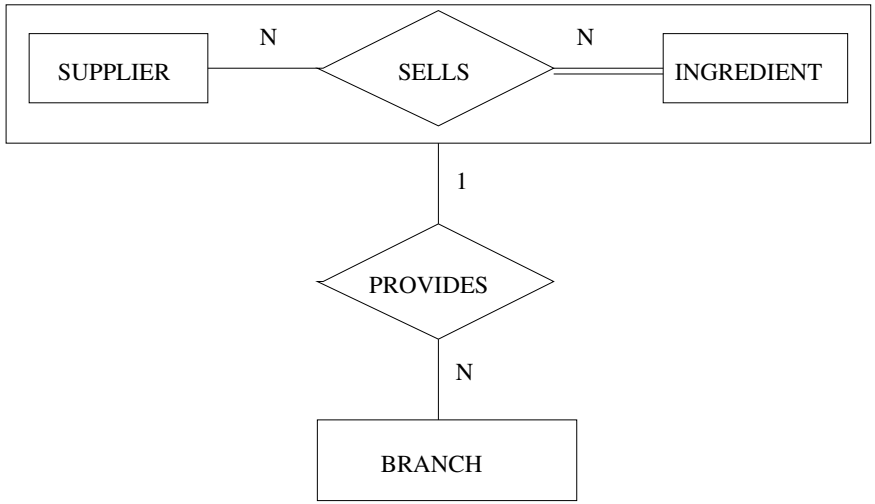
EER model. Aggregations

- The standard ER model can only set relationships between entity types. We cannot connect a relationship with another relationship directly.
- However, there are some cases in which we need to deal with a relationship **as if it were an entity type at a higher level.**
- The EER model can treat a set of componentes (several entities and a relationship that connect them all) **as if they were a single entity type.**
- It is represented in an EER diagram enclosing that set in a rectangle, and connecting that rectangle to other relationships in the EER diagram.

EER model. Aggregations. Example

- Let us assume that in the example of the food delivery company we have a relationship `sells` that represents the ingredients that a supplier sells.

- If we are told that *"a branch is provided by a supplier* **of those ingredients that the supplier sells**", the ternary relationship `provides` seen before can now relate the branch with the aggregation `supplier:ingredient`:



From conceptual design to logical design

- DBMS do not implement the ER model directly.
- The conceptual design produced by the ER model must be converted to a lower-level logical design, the **relational model.**
- This is a systematic process following a series of rules that will study in the next lesson.
- Several CASE tools and frameworks generate (semi-)automatically the DDL schema of a DB from the ER model (or the UML class diagrams).