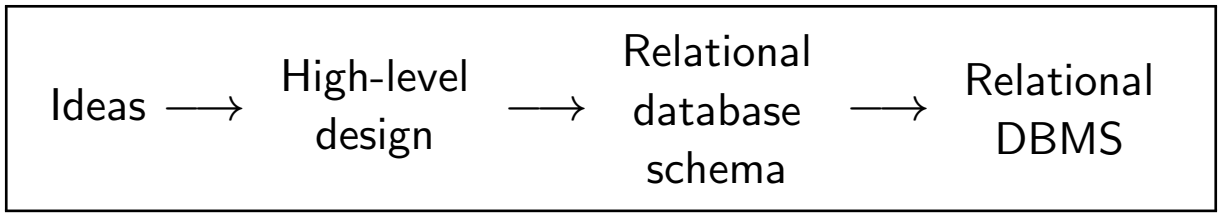
2. Conceptual Modeling using the Entity-Relationship Model

Contents

- Basic concepts: entities and entity types, attributes and keys, relationships and relationship types
- Entity-Relationship schema (aka ER diagram)
- Constraints on relationship types
- Design choices
- Enhanced Entity-Relationship model features
- Steps in designing an ER schema
- Translation of an ER schema to tables

What does Conceptual Design include?



- Entity-Relationship model is used in the conceptual design of a database (☞ conceptual level, conceptual schema)
- Design is independent of all physical considerations (DBMS, OS, . . .).
  Questions that are addressed during conceptual design:
  - What are the entities and relationships of interest (*mini-world*)?
  - What information about entities and relationships among entities needs to be stored in the database?
  - What are the constraints (or business rules) that (must) hold for the entities and relationships?
- A database schema in the ER model can be represented pictorially
  (*Entity-Relationship diagram*)

Entity Types, Entity Sets, Attributes and Keys

- *Entity:* real-world object or thing with an independent existence and which is distinguishable from other objects. Examples are a person, car, customer, product, gene, book etc.

- *Attributes:* an entity is represented by a set of attributes (its descriptive properties), e.g., name, age, salary, price etc. Attribute values that describe each entity become a major part of the data eventually stored in a database.

- With each attribute a *domain* is associated, i.e., a set of permitted values for an attribute. Possible domains are INTEGER, STRING, DATE, etc.

- *Entity Type:* Collection of entities that all have the same attributes, e.g., persons, cars, customers etc.

- *Entity Set:* Collection of entities of a particular entity type at any point in time; entity set is typically referred to using the same name as entity type. instance

Key attributes of an Entity Type

- Entities of an entity type need to be distinguishable.
- A *superkey* of an entity type is a set of one or more attributes whose values uniquely determine each entity in an entity set.
- A *candidate key* of an entity type is a minimal (in terms of number of attributes) superkey.
- For an entity type, several candidate keys may exist. During conceptual design, one of the candidate keys is selected to be the *primary key* of the entity type.

Relationships, Relationship Types, and Relationship Sets

- *Relationship (instance):* association among two or more entities, e.g.,
  "customer 'Smith' orders product 'PC42' "

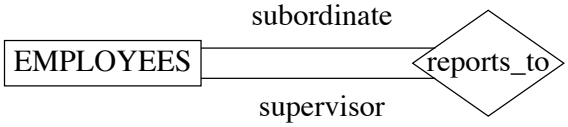
- *Relationship Type:* collection of similar relationships
  An n-ary relationship type R links n entity types $E_1, \ldots, E_n$.
  Each relationship in a relationship set R of a relationship type involves entities $e_1 \in E_1, \ldots, e_n \in E_n$

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

  where $(e_1, \ldots, e_n)$ is a relationship.

- *Degree of a relationship:* refers to the number of entity types that participate in the relationship type (binary, ternary, . . .).

- *Roles:* The same entity type can participate more than once in a relationship type.



  Role labels clarify semantics of a relationship, i.e., the way in which an entity participates in a relationship.
  $\rightsquigarrow$ *recursive relationship*.

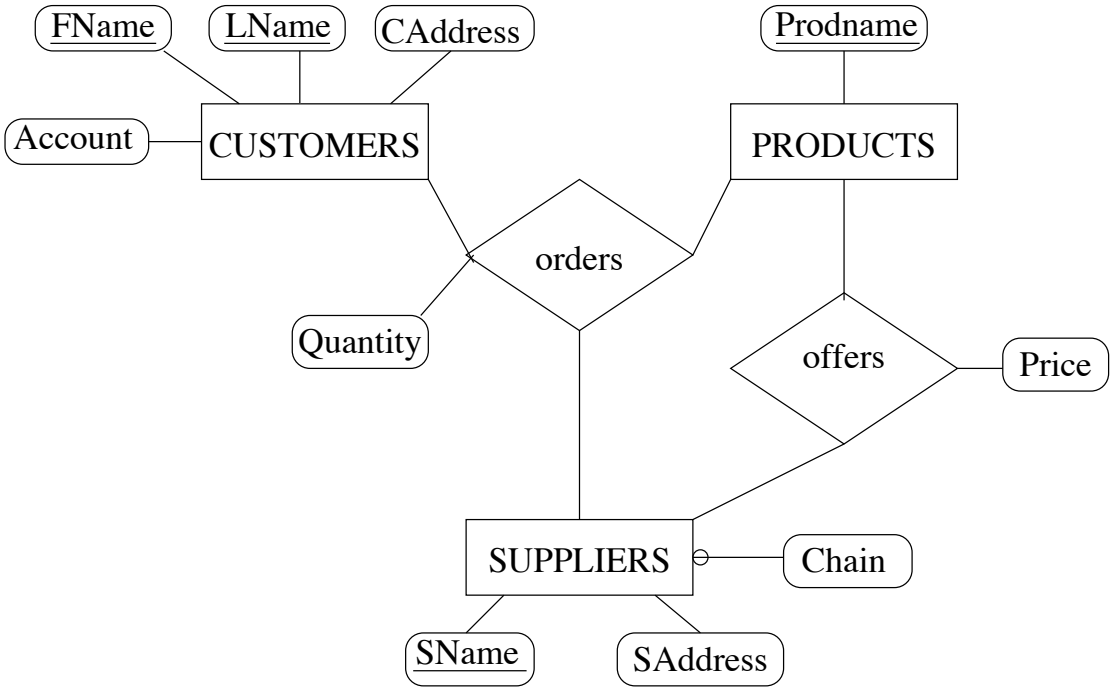
- *Relationship Attributes:* A relationship type can have attributes describing properties of a relationship.

  "customer 'Smith' ordered product 'PC42' on January 11, 2005, for $2345".

  These are attributes that cannot be associated with participating entities only, i.e., they make only sense in the context of a relationship.

- Note that a relationship does not have key attributes! The identification of a particular relationship in a relationship set occurs through the keys of participating entities.

Example of an Entity-Relationship Diagram



Customers-Suppliers-Products Entity-Relationship Diagram

- **Rectangles** represent entity types
- **Ellipses** represent attributes
- **Diamonds** represent relationship types
- **Lines** link attributes to entity types and entity types to relationship types
- **Primary key** attributes are underlined
- **Empty Circle** at the end of a line linking an attribute to an entity type represents an optional (null) attribute (not mentioned in textbook)

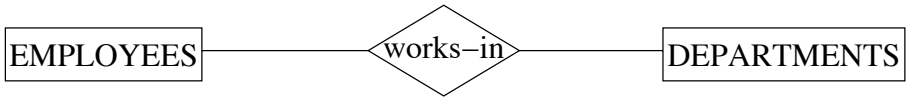
  Not in the above diagram, but later in examples:
- **Double Ellipses** represent multi-valued attributes

Constraints on Relationship Types

Limit the number of possible combinations of entities that may participate in a relationship set. There are two types of constraints: *cardinality ratio* and *participation constraints*

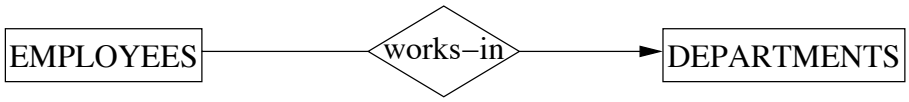
Very useful concept in describing binary relationship types. For binary relationships, the cardinality ratio must be one of the following types:

- *Many-To-Many* (default)



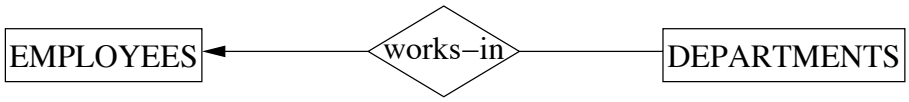
  Meaning: An employee can work in many departments (≥ 0), and a department can have several employees

- *Many-To-One*



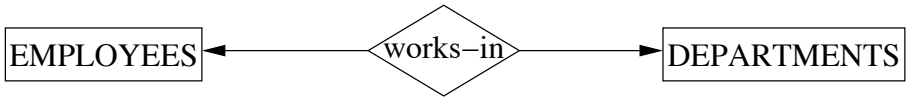
  Meaning: An employee can work in at most one department (≤ 1), and a department can have several employees.

- *One-To-Many*



Meaning: An employee can work in many departments (≥ 0), but a department can have at most one employee.

- *One-To-One*



Meaning: An employee can work in at most one department, and a department can have at most one employee.

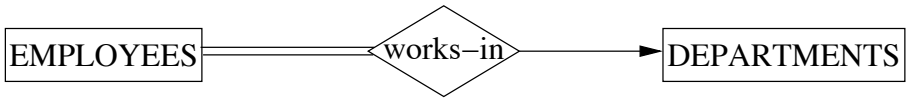
Constraints on Relationship Types (cont.)

A many-one relationship type (and the counterpart one-many) is also often called a *functional relationship*.

Cardinality ratio of a relationship can affect the placement of a relationship attribute. E.g., in case of a many-one relationship type, one can place a relationship attribute at a participating entity type.

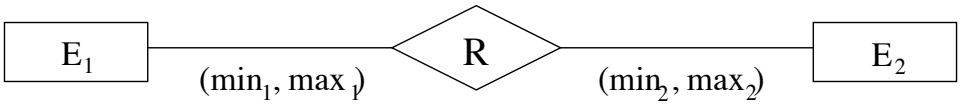
Participation constraint: specifies whether the existence of an entity $e \in E$ depends on being related to another entity via the relationship type R.

- total: each entity $e \in E$ must participate in a relationship, it cannot exist without that participation (total participation aka existence dependency).



- partial: default; each entity $e \in E$ can participate in a relationship

Instead of a cardinality ratio or participation constraint, more precise *cardinality limits* (aka *degree constraints* in textbook) can be associated with relationship types:



Each entity $e_1 \in E_1$ must participate in relationship set R at least $\min_1$ and at most $\max_1$ times (analogous for $e_2 \in E_2$).

Frequently used cardinalities

Relationship	$(\min_1, \max_1)$	$(\min_2, \max_2)$	pictorial notation
many-to-many	$(0, *)$	$(0, *)$	
many-to-one	$(0, 1)$	$(0, *)$	
one-to-one	$(0, 1)$	$(0, 1)$	

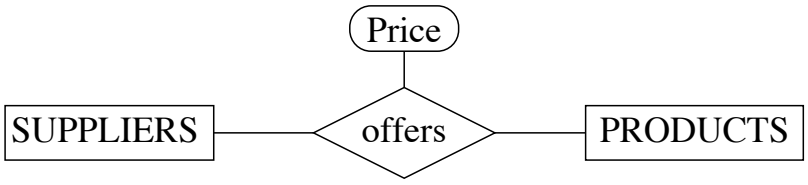
Design Choices for ER Conceptual Design

It is possible to define entities and their relationships in a number of different ways (in the same model!).

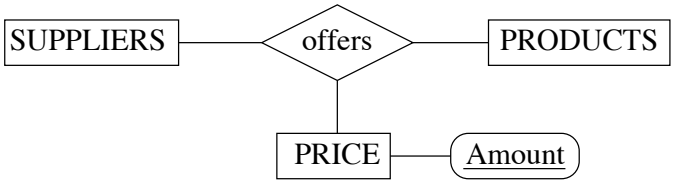
- Should a real-world concept be modeled as an entity type, attribute, or relationship type?

  Is "Address" an attribute or an entity type? Decision depends upon the use one wants to make of address information. If one is interested in the structure, e.g., (City, Street, Zip-Code), Address must be modeled as an entity type (or as a *complex attribute*).

- Should a concept be modeled as an entity type or relationship type?



Here a supplier cannot offer the same product for different prices! Why?
Modeling price as an entity type resolves this problem:

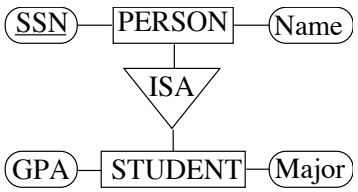


Enhanced ER Modeling Concepts

Although most properties of entities and relationships can be expressed using the basic modeling constructs, some of them are costly and difficult to express (and to understand). That's why there are some extensions to the ER model.

Subclasses, Superclasses, and Inheritance

- In some cases, an entity type has numerous subgroupings of its entities that are meaningful and need to be represented explicitly because of their significance to the DB application.

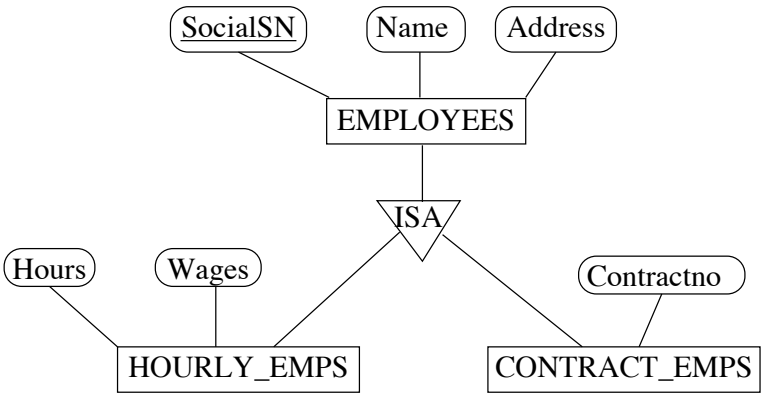


- Relationships and attributes of superclass are inherited to subclass (in particular primary key attribute(s)); subclass can have additional attributes and relationships

- An entity cannot exist merely by being a member of only a subclass.

Specialization

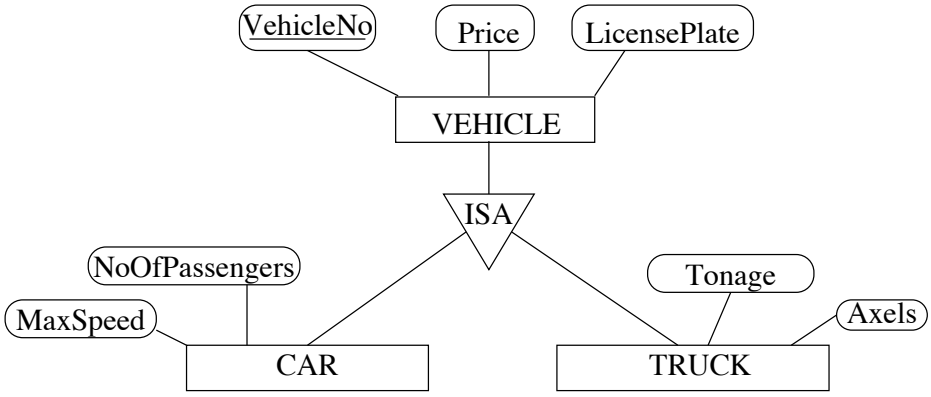
- Process of defining a set of subclasses of an entity type (top-down)



HOURLY_EMPS is a subclass of EMPLOYEES and thus inherits its attributes and relationships (same for CONTRACT_EMPS).

Generalization:

- Reverse process of specialization (bottom-up); identify common features of entity types and generalize them into single superclass (including primary key!)



Constraints on Specialization

- disjointness and totality constraints

  disjoint, total/partial: each entity in the superclass must/can be in exactly one subclass

  The disjointness constraint is indicated by the word "disjoint" right next to the ISA triangle

  The totality constraint is indicated by double lines leading from the superclass to the ISA triangle

- overlapping constraints

  overlapping, total (. . . must be in at least one subclass) In this case, only double lines leading to the ISA triangle are used.

  overlapping, partial (. . . can be in at least one subclass)

- Note: special rules are required to propagate deletions from superclass/subclass (implemented later).

Note that for generalization, each entity in the superclass must belong to exactly one subclass

Steps in Designing an Entity-Relationship Schema

[Step 1] Identify entity types (entity type vs. attribute)

[Step 2] Identify relationship types

[Step 3] Identify and associate attributes with entity and relationship types

[Step 4] Determine attribute domains

[Step 5] Determine primary key attributes for entity types

[Step 6] Associate (refined) cardinality ratio(s) with relationship types

[Step 7] Design generalization/specialization hierarchies including constraints (includes natural language statements as well)

Translation of ER Schema into Tables

- An ER schema can be represented by a collection of tables which represent contents of the database (instance).
- Primary keys allow entity types and relationship types to be expressed uniformly as tables.
- For each entity and relationship type, a unique table can be derived which is assigned the name of the corresponding entity or relationship type.
- Each table has a number of columns that correspond to the attributes and which have unique names. An attribute of a table has the same domain as the attribute in the ER schema.

- Translating an ER schema into a collection of tables is the basis for deriving a relational database schema from an ER diagram.

Translating Entity Types into Tables

- Given an entity type E_1 with (atomic) attributes $A_1, \ldots, A_n$ and associated domains $D_1, \ldots, D_n$.
- Translation of the entity type CUSTOMERS into table CUSTOMERS

FName	LName	CAddress	Account
Michael	Smith	San Francisco	10,000
George	Jones	New York	2,000
Kent	Clark	Los Angeles	-4,600
...	...	...	...

In SQL:

```
create table Customers(
 FName char(40),
 LName char(40),
 CAddress char(70),
 Account real
);
```

- A row in such a table corresponds to an entity from the entity set.

Translating Relationship Types into Tables

- A many-many relationship type is represented as a table with columns for the primary key attributes of the participating entity types, and any descriptive attributes of the relationship type.

  Example: Relationship type offers

Prodname	SName	Price
PC42	Hal-Mart	2,100
MacIV	Sears	2,500
. . .	. . .	. . .

  Prodname and SName are the primary key attributes of the entity types SUPPLIERS and PRODUCTS.

- Translation of one-many and many-one (functional) and one-one relationship types into tables can be optimized
  $\leadsto$ no table for relationship type necessary!

Translating Subclasses/Superclasses into Tables

- Method 1: Form a table for the superclass and form a table for each subclass. Include the primary key attributes of the superclass in each such table.

  Example:

  ```
  Employees(SocialSN, Name, Address)
  Hourly_Emps(SocialSN, Hours, Wages)
  Contract_Emps(SocialSN, ContractNo)
  ```

- Method 2: Form a table for each subclass and include all attributes of the superclass.

  ```
  Hourly_Emps(SocialSN, Name, Address, Hours, Wages)
  Contract_Emps(SocialSN, Name, Address, Contractno)
  ```

  Method 2 has no table for the superclass EMPLOYEES.

- Method 3: Use `null` values

  ```
  Employees(SocialSN, Name, Address,
            Hours, Wages, ContractNo)
  ```

  Hourly employees will have a `null` value for `ContractNo`. Contract employees will have `null` values for `Hours` and `Wages`.

Summary of Conceptual Design

- Conceptual design follows requirements analysis, yields a high level description of data to be stored (conceptual level).
- ER model is a popular model for conceptual design, constructs are expressive, close to the way people think about applications; supported by many CASE tools.
- Basic constructs are *entities*, *relationships*, and *attributes*
- Some additional constructs: *ISA hierarchies*, *cardinality ratios*, . . .
- There are many variations on ER model constructs .
- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *structural constraints*, *constraints on specializations*
  - Some of them can be expressed in SQL when translating entity and relationship types into tables
  - Not all constraints can be expressed in the ER model
  - Constraints play an important role in determining a good database design for an application domain.
- ER design is *subjective*: There are many ways to model a given scenario! Analyzing alternative schemas is important! Entity type vs. attribute, entity type vs. relationship type, binary vs. n-ary relationship type, use of IS-A, generalization and specialization, . . .
- Ensuring a good database design includes analyzing and further refining relational schema obtained through translating ER schema.