Generalization

- goals
  - abstraction at the set level: better (i.e., more understandable and more concise) structuring of entity sets
  - abstraction at the instance level: similar entities are to be modeled by a common entity set

- „factoring“ (extracting) properties (attributes, relationships) of similar entity sets (subclass, subtypes, categories) to a common superclass (supertype)

- properties that cannot be extracted remain with the respective subclass, i.e., the subclass is a specialization of the superclass

- inheritance as the key concept of generalization: a subclass inherits all properties of a superclass
entities of a subclass are implicitly considered as entities of the superclass, therefore **is-a** in the graphical representation

→ set of entities of the subclass is a subset of the set of entities of the superclass

two special cases

- **disjoint/overlapping specialization**: all subclasses of a superclass are pairwise disjoint/overlapping

- **total specialization**: the superclass does not contain explicit elements, but is only given by the union of its subclasses (antonym: **partial specialization**)

```
name

university-members

is-a

reg-id

students

employees

pers-id

is-a

research area

assistants

professors

rank

room
```
Aggregation

- goal: distinct entity sets which together form a structured superclass are associated with each other
- an **aggregation** is a special relationship set which associates each superior entity set with several subordinate entity sets
- **part-of**-relationship
- example: construction of a bicycle
4. Relational Data Model

4.1 Introduction


- commercial DBMSs like Oracle, Informix, SQL Server, Sybase, DB/2 are based on the relational model

- reasons for the success of the relational data model
  - flat tables (relations) as the simple underlying data structure
  - no nested complicated structures
  - set oriented processing of data in contrast to record oriented processing prevailing until then (hierarchical model, network model)
  - simple comprehensibility also for the unskilled user
  - good performance for standard database applications
  - existence of a mature, formal theory (in contrast to other data models), in particular with respect to the design of relational databases and with respect to an efficient processing of user queries
4.2 Definition of the Relational Model

Basic structure

- Given \( n \) domains \( D_1, D_2, ..., D_n \)
  - examples for domains: data types \textit{integer}, \textit{string}[20], \textit{real}, \textit{bool}, \textit{date}, ...
  - domains need not be disjoint, i.e., \( D_i = D_j \) is admissible for \( i \neq j \)
  - domains may contain only \textit{atomic} values, they must not be structured

- a \textit{relation (instance)} \( r_R \) is defined as a subset of the Cartesian product of \( n \) domains:
  \[
  r_R \subseteq D_1 \times D_2 \times ... \times D_n \quad (r_R \text{ finite})
  \]

- \( r_R \) is an \textit{occurrence (instance)} of a pertaining \textit{relation schema} \( R \) (analogously to the programming language notions of \textit{variable} and \textit{type}).

- an element of the set \( R \) is called \textit{tuple}, tuple has \textit{arity} \( n \)

- example:
  - domains: \( D_1 = \{a, b, c\}, D_2 = \{0, 1\} \)
  - Cartesian product: \( D_1 \times D_2 = \{(a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1)\} \)
  - examples for instances: \( r_1 = \{(a, 0), (b, 0), (c, 0), (c, 1)\}, r_2 = \{(a, 0)\}, r_3 = \emptyset \)
Some basic mathematical concepts

- How is the *subset* relationship ("\(\subseteq\)") formally defined?
  Given two sets \(A\) and \(B\). Then \(A \subseteq B \iff \) ?

- How is the *cross product* ("\(\times\)") formally defined?
  Given two sets \(A\) and \(B\). Then \(A \times B = \) ?

- How many elements does \(A \times B\) have?

- What is a *relation* then?

- What is the difference between a *relation* and a *function*?
Schema definition

- distinction between the **schema** of a relation $R$, which is given by the $n$ domains (data types), and the current **instance** of this relation schema, which is given by a subset of the Cartesian product.

- schema analogously to the programming language notion of type definition.

- a relation schema $R$, denoted by $R(A_1, A_2, ..., A_n)$, consists of the relation name $R$ and a list of attributes $A_1, A_2, ..., A_n$.

- each attribute $A_i$ is the name of a role played by domain $D_i$ in the relation schema $R$.
  - $D_i$ is also the domain (type) of $A_i$.
  - notation: $D_i = \text{dom}(A_i)$.

- for the schema $R(A_1, A_2, ..., A_n)$ holds: $r_R \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n)$.

- we describe the schema of $R$ also in the form $R(A_1 : D_1, A_2 : D_2, ..., A_n : D_n)$.

- because we often do **not** make a clear distinction between the meta level (schema) and the instance level (occurrence), we also denote relation instances with the letter $R$. 
A representation of a relation as tables with **rows** (tupels) and **columns**

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

A and $N$ are attributes and have the function of column names

---

**example: relation Students(RegNo : string, Name : string, Age : integer, ...)**

<table>
<thead>
<tr>
<th>Students</th>
<th>RegNo</th>
<th>Name</th>
<th>Age</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Meyer</td>
<td>John</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>456123</td>
<td>Smith</td>
<td>Ben</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>321654</td>
<td>Benson</td>
<td>Jeff</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>654321</td>
<td>Bates</td>
<td>Allen</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>