Logical data model

Versus technical data model implementation

Table of content

Logical data model definition

A logical data model in systems engineering is a representation of an organization's data, organized in terms of entities and relationships and is independent of any particular data management technology …

Wikipedia

… and technical implementation

Logical data model advantages

Reasons for building a logical data model:

- Helps common understanding of business data elements and requirements
- Provides foundation for designing a database
- Facilitates avoidance of data redundancy and thus prevent data & business transaction inconsistency
- Facilitates data re-use and sharing
- Decreases development and maintenance time and cost
- Confirms a logical process model and helps impact analysis.

Modeling benefits:

- Facilitates business process improvement
- Focuses on requirements independent of technology
- Facilitates data re-use and sharing
- Increases return on investment
- Centralizes metadata
- Fosters seamless communication between applications
- Focuses communication for data analysis and project team members
- Establishes a consistent naming scheme

Wikipedia

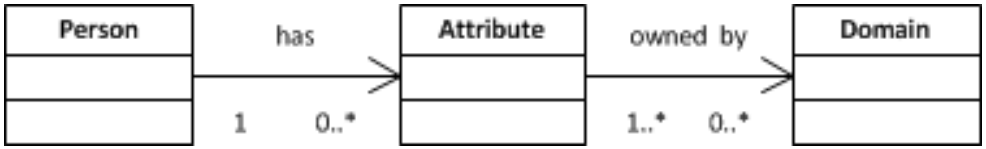
Data model layers

- Physical layer/view
  - Displays technical artifacts not meaningful to the business
  - All tables and fields are visible:
    - Mapping tables, …
  - Potentially accessed natively for e.g. maintenance, backup, replication purposes
- Logical layer/view
  - Hides database artifacts not meaningful to the business users/applications
  - Accessed by business applications using Structured Query Language (SQL)
- Stored procedures
  - Stored procedures are sub-routines stored within the database and invoked by business applications
  - Stored procedures hide/encapsulate the entire technical/logical database model
  - Stored procedures implement atomic business functions
  - They belong logically to the 3-tier "logic" layer although technically implemented in the database

Example

Let's assume that we want to get the status of a person in the scope of the "Insurability" business domain.

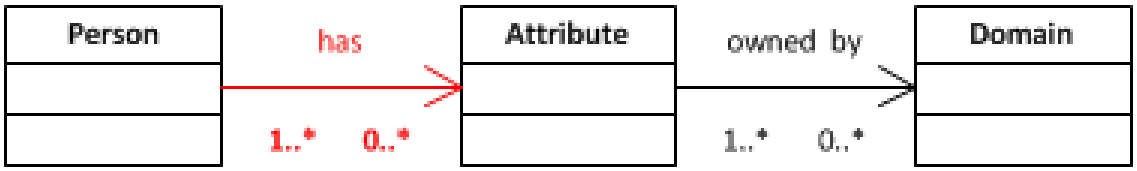
- In the database, a person is a resource "shared" by more than one business domain with potentially domain specific attributes e.g. An insurability flag in the scope of the insurability domain



- A person can have "common" and "domain specific" attributes.
- The specific attributes must be accessible only by its rightful owner (business domain) and associated users (business actor involved in this business domain)

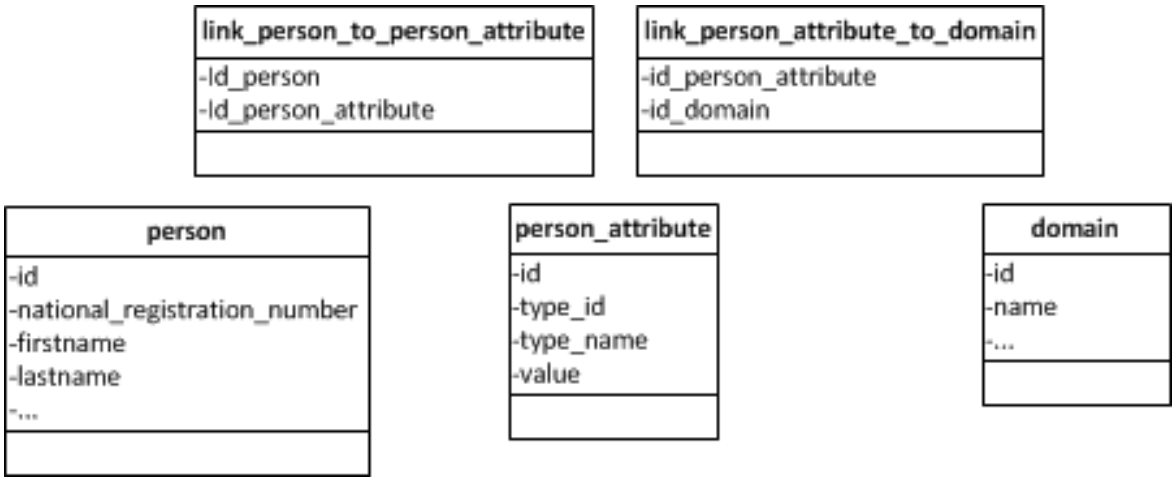
Example : technical implementation

- Practically, several persons may also share the same attribute value, e.g. "Marital status" = "bachelor"
- For technical implementation reasons it might be a good idea to store all attributes (common and specific) in the same tables instead of maintaining two separate tables
- The technical implementation would then become:



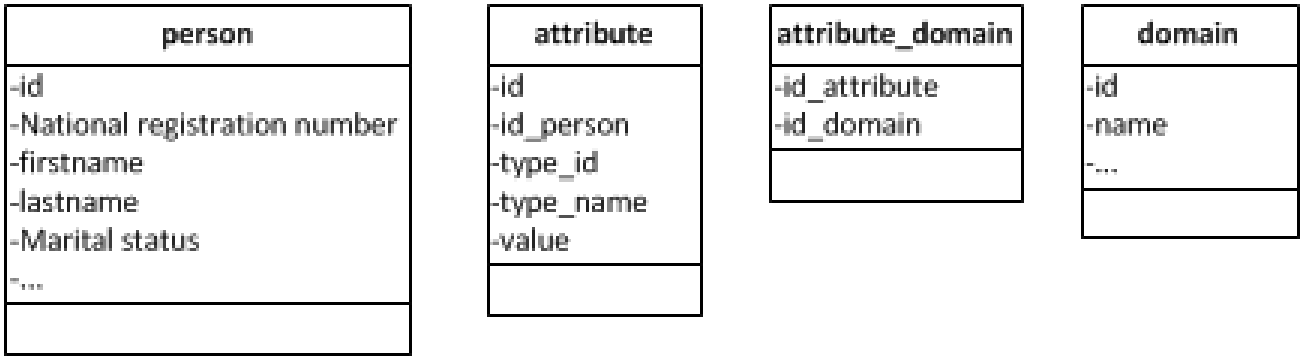
Example : technical implementation (cntd)

- The cardinality between person and attribute becomes a many-to-many relationship and requires and additional intermediary table which has no business meaning.
- Also, the information about the business domain owning the data can be externalized because:
  - It is shared by other "objects" in the database
  - It has additional attributes not meaningful in the scope of a "person"
  - …



Example : logical data model

The corresponding logical data model could be:

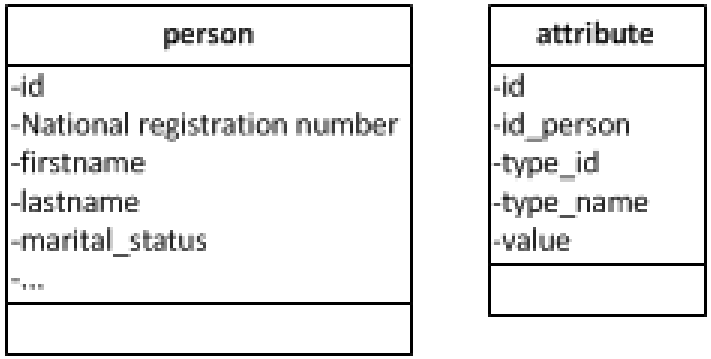


- Technical implementation choices (mapping table, …) are not visible
- "shared" person attributes are shown "inside" a person object/record e.g. "Marital status".
- A standard naming scheme is exposed

Example : logical data model

For a specific business domain

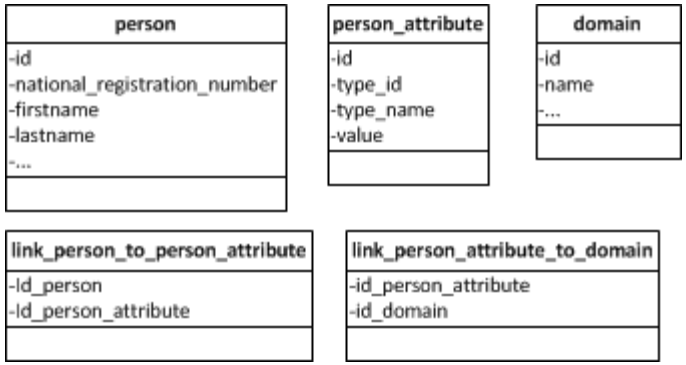
- A specific view on a logical data model can be provided for a specific business domain (data owner)
- The owner of the data can see only the data he owns (in addition to common data):



- Filtering is done on the owner (domain) of the attributes:
  - Show all attributes not linked to a domain: the shared/common ones
  - Show all attributes for e.g. the "Insurability" domain

Example : technical implementation SQL

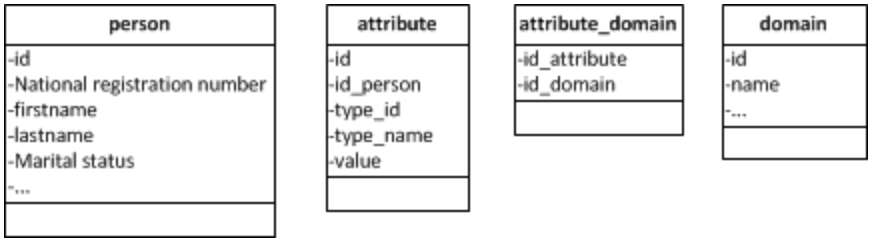
- E.g. Get the insurability status of Robert Smith:




```
SELECT
  person_attribute.value
FROM
  person INNER JOIN
    link_person_to_person_attribute INNER JOIN
      person_attribute INNER JOIN
        link_person_attribute_to_domain INNER JOIN
          domain
        ON link_person_attribute_to_domain.id_domain = domain.id
      ON person_attribute.id = link_person_attribute_to_domain.id_person_attribute
    ON link_person_to_person_attribute.id_person_attribute = person_attribute.id
  ON person.id = link_person_to_person_attribute.id_person
WHERE
  person.firstname = "Robert",
  person.lastname = "Smith",
  person_attribute.type_id = 5,    ' 5 => "insurability status" = person_attribute.type_name
  domain.id = 1                    ' 1 => "Insurability" = domain.name
```

Example : logical data model SQL

E.g. Get the insurability status of Robert Smith:




```
SELECT
  attribute.value
FROM
  person INNER JOIN
    attribute INNER JOIN
      attribute_domain INNER JOIN
        domain
      ON attribute_domain.id_domain = domain.id
    ON attribute.id = attribute_domain.id_attribute
  ON person.id = attribute.id_person
WHERE
  person.firstname = "Robert",
  person.lastname = "Smith",
  attribute.type_id = 5,        ' 5 => “insurability status” = person_attribute.type_name
  domain.id = 1                 ' 1 => “Insurability” = domain.name
```

Example : stored procedures

- The corresponding stored procedure could be

  GetInsurabilityByName(firstname, lastname) or
  GetInsurabilityByRegistrationNumber(national_registration_number)

- Stored in the database itself and hides the physical and logical data model complexity to the calling application.
- Part of the business logic (business sub-functions/sub-routines) are stored in the database.

End of presentation

Questions