Data and Process Modelling

4. Relational Mapping

Marco Montali

KRDB Research Centre for Knowledge and Data
Faculty of Computer Science
Free University of Bozen-Bolzano

A.Y. 2014/2015

From Design to Implementation

- A conceptual schema is designed to ultimately store, update and query the relevant information of a domain.
- Concrete typical outcomes:
  - Development of a *physical database schema*.
  - Development of the *model layer* of an (object-oriented) software application.
- In both cases, we need to *map* the conceptual schema to a corresponding logical, and then physical, schema.
- Main methodological steps:
  1. Design the conceptual schema.
  2. Annotate the conceptual schema with mapping choices.
  3. Mapping the conceptual schema to a logical schema (relational, object-oriented, ...).
  4. Manipulate the logical schema.
  5. Generate the corresponding physical schema (MySQL DB, Java classes, ...).

From Conceptual to Relational Schema

- Relational model (Codd, 1969): connection between databases and first-order logic.
- Database schema constituted by a set of relation schemas containing:
  - name of the relation (*table name*);
  - (named) set of attributes (*table columns*), each ranging over some data domain (unnamed equivalent version also exists).
- Extensional information represented as a set of unnamed tuples (*records*) over such relations, where each attribute is filled in with a value belonging to the corresponding data domain.
- Each tuple maintains information about one or more fact types.
  - Elementary fact types (e.g., Student with code "1234" attends the Course named "Conceptual Modeling").
  - Existential fact types (e.g., there exists a Course named "Conceptual Modeling").
- In addition:
  - Constraints, in particular *keys*, *foreign keys*, *optionality*.
  - Derivation rules.

Data Domains

- Relational model supports semantic domains for attributes.
- SQL standard now supports user-defined types for attributes
  $\rightarrow$ account for semantic domains.
- However, many commercial tools still provide support only for primitive data types (strings, numbers, datetime, bit/boolean, ...).
- Furthermore, primitive data types facilitate the interoperation with other languages/environment (see JDBC conversion in Java and object-relational mapping).

We will *abstract away* from data domains in our analysis.

It is of *fundamental* importance to decide how to *map* semantic domains to attribute types, and *track* the choice. If only primitive types are used, then semantic domains translate into *data validation policies* (external constraints).

Relational Schemas and Layouts

No industrial standard for representing relational schemas.

- Horizontal layout:

  Employee(empNr, empName, deptCode, gender, salary, tax)

- Vertical layout:

Employee
empNr
empName
deptCode
gender
salary
tax

Relational Schemas and Internal Uniqueness Constraints

Internal UCs provide candidate keys for identifying tuples in a relation.

- Key: minimal set of uniquely constrained attributes.
  - Horizontal layout: underlined attributes (as in ORM).
  - Vertical layout: "Uk" decoration for the attributes involved in UC identified by number "k".
- Primary key: preferred key for identification.
  - Horizontal layout: only existing UC $\rightarrow$ underlined attributes; other alternative keys $\rightarrow$ doubly underlined attributes.
  - Vertical layout: "PK" decoration for involved attributes, which are also underlined.

Employee(empNr, empName, deptCode, gender, salary, tax)

Employee	
PK	empNr
U1	empName
U1	deptCode
	gender
	salary
	tax

Relational Schemas and Mandatory Constraints

Mandatory constraint on a column: the column does not allow *null* values.
Contrariwise: optional column.

- Horizontal layout: square brackets around optional columns.
- Vertical layout: boldface around mandatory columns or square brackets around optional columns (for diagrams written on paper).

Employee(empNr, empName, deptCode, gender, salary, [tax])

Employee	
PK	**empNr**
U1	**empName**
U1	**deptCode**
	gender
	salary
	tax

Basic Integrity Rule

A primary key contains no null: all its constitutive columns are mandatory.

Correspondence with Mandatory Roles in ORM

Problem: each tuple in a relation expresses one or more ORM fact, while different facts about the same object can be stored in different tables.

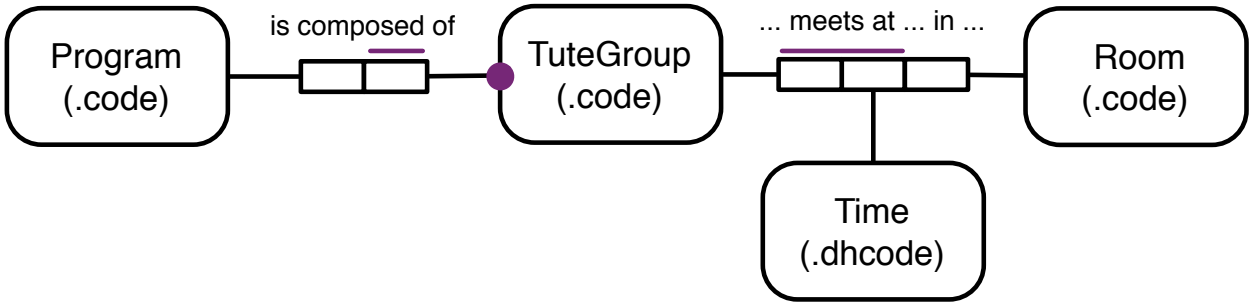
- How to map a mandatory ORM role to the relational schema?
- Remember: role r mandatory for object type O if for each other r' played by O we have $pop(r') \subseteq pop(r) \rightarrow$ *implied* subset constraint from r' to r.
- Mapping in the relational schema:
  1. Column corresponding to r marked as mandatory.
  2. Subset constraints from the column corresponding to r' in another table to the column of r. This is a referential integrity constraint, and the r' column is a foreign key referencing the r column.

  This holds for every role r', and can easily be generalized to combination of columns for extended mandatory constraints.

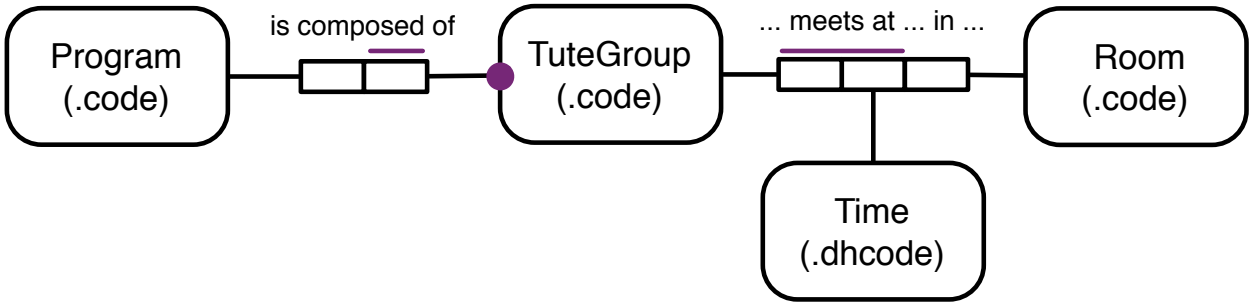
Basic integrity rule

Referential integrity: every nonnull value of a foreign key must match one of the values of the referred primary key.

Foreign Keys and Layouts



Foreign Keys and Layouts



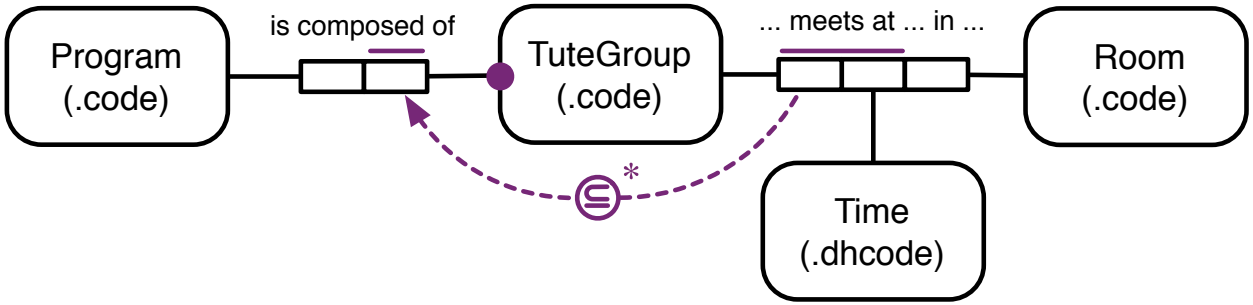
TuteGroup(tuteCode, progCode)

Meets(tuteCode, timeDHCode, roomCode)

TuteGroup	
PK	**tuteCode**
	progCode

Meets	
PK	**tuteCode**
PK	**timeDHCode**
	roomCode

Foreign Keys and Layouts



TuteGroup(tuteCode, progCode)

Meets(tuteCode, timeDHCode, roomCode)

TuteGroup	
PK	**tuteCode**
	progCode

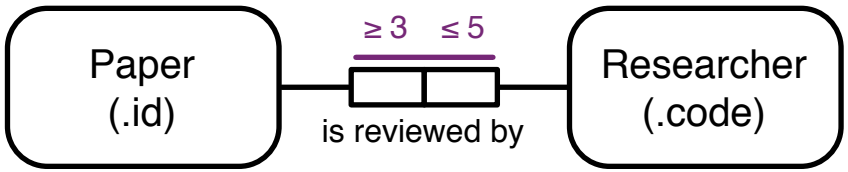
Meets	
PK, FK1	**tuteCode**
PK	**timeDHCode**
	roomCode

Other Constraints

- As we will see, many ORM constraints are mapped to keys and references, depending on the context.
- Other constraints are maintained as annotations on the horizontal layout, following the ORM conventions.

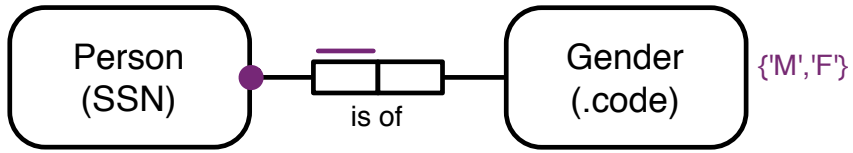
Other Constraints

- As we will see, many ORM constraints are mapped to keys and references, depending on the context.
- Other constraints are maintained as annotations on the horizontal layout, following the ORM conventions.



Reviewed(paperId, resCode)

$\geq 3 \qquad \leq 5$



Person(SSN, genderCode)

$\{'M','F'\}$

- Some of such constraints can be then enforced as SQL constraints, depending on the version and on the environment.
- Example:

```
check(not exists
        (select resCode from Reviewed
         group by resCode having count(*) > 5))
```

Derivation Rules and Relational Schemas

Several options exist to map derivation rules to the relational level (again, support and language change with the DBMS at hand):

1. View that contains the derived facts, calculating them using the derivation rules.
2. Generated column that, whenever a tuple is added/updated, is automatically fed with a value calculated using the derivation rules.
3. Triggered column, using a trigger that encapsulates the algorithmic logic of the derivation rules.
4. Stored procedure, run when needed to update the information base so as to calculate the derived information using the fresh values.

Relational Mapping Algorithm: Rmap

Translation of a conceptual schema to a relational one balancing between

- *efficiency* (less tables);
- *redundancy avoidance*, no repetition of primitive facts (more tables).

Relational Mapping Algorithm: Rmap

Translation of a conceptual schema to a relational one balancing between

- *efficiency* (less tables);
- *redundancy avoidance*, no repetition of primitive facts (more tables).

Rmap contains strategy for the first issue, and guarantees the second thanks to the following principle.

Redundancy Avoidance

Each fact type of the conceptual schema is mapped to one table, so that its instances appear only once.

How to enforce this principle?

Redundancy Avoidance Strategies

1. Fact type with a compound internal UC → mapped to dedicated table, using the UC as PK.
2. Fact types with functional roles on the same object type → grouped into a unique table, with the object type's preferred identifier as PK.

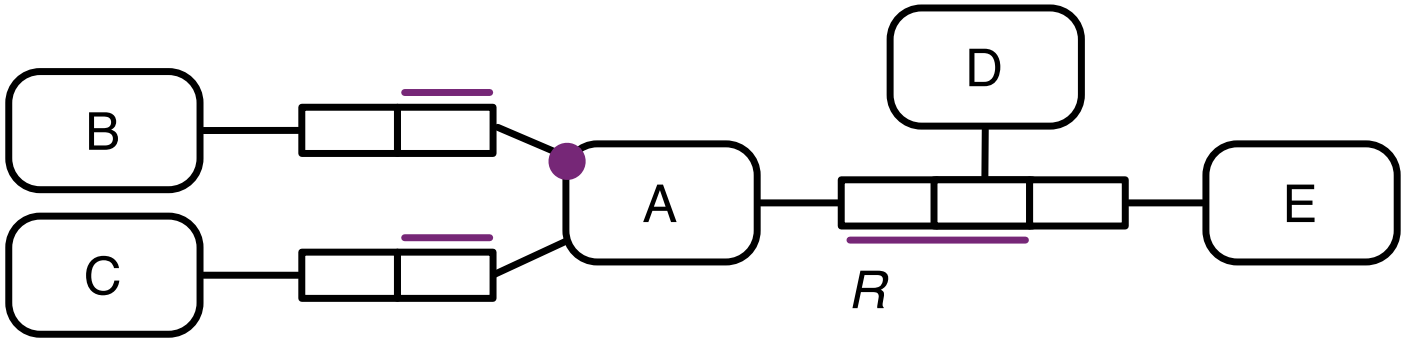
Object types are mapped to one or more attributes, depending on their preferred identification scheme.

Naming guidelines:

- Fix table names using the reading of the fact type (case 1.) or the name of the object type (case 2.).
- Object types' columns are named by adding in front of their name a prefix corresponding to the object type name they refer to (e.g., EmpCode).
- Use informative names for columns, using, if possible, the same name for FKs and their references.
- If a predicate contains roles played by the same object type, use the name of the roles (e.g., superpart and subpart).

Prototypical Examples

Hp: a, b, c, d, e represent the set of attributes representing the preferred identification scheme for the corresponding object type.



Case 2

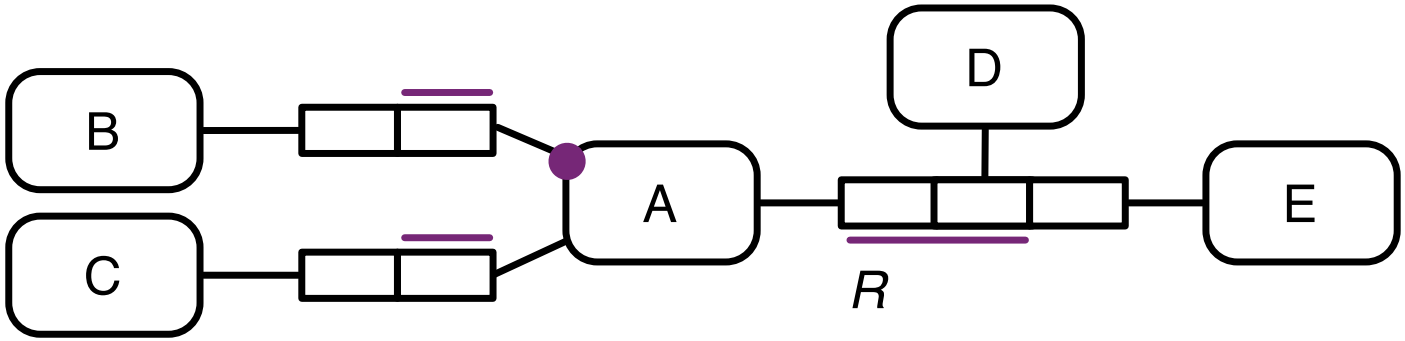
- Applies to binary fact types with simple UCs all over roles played by the same object type.
- Mandatory dots determine mandatory columns in the grouped table.

Case 1

- Applies to binary fact types with spanning UCs and to n-ary fact types with n $>$ 2 (why?).
- FKs must be added for those roles attached to object types that play mandatory roles elsewhere ($\subseteq$).

Prototypical Examples

Hp: a, b, c, d, e represent the set of attributes representing the preferred identification scheme for the corresponding object type.



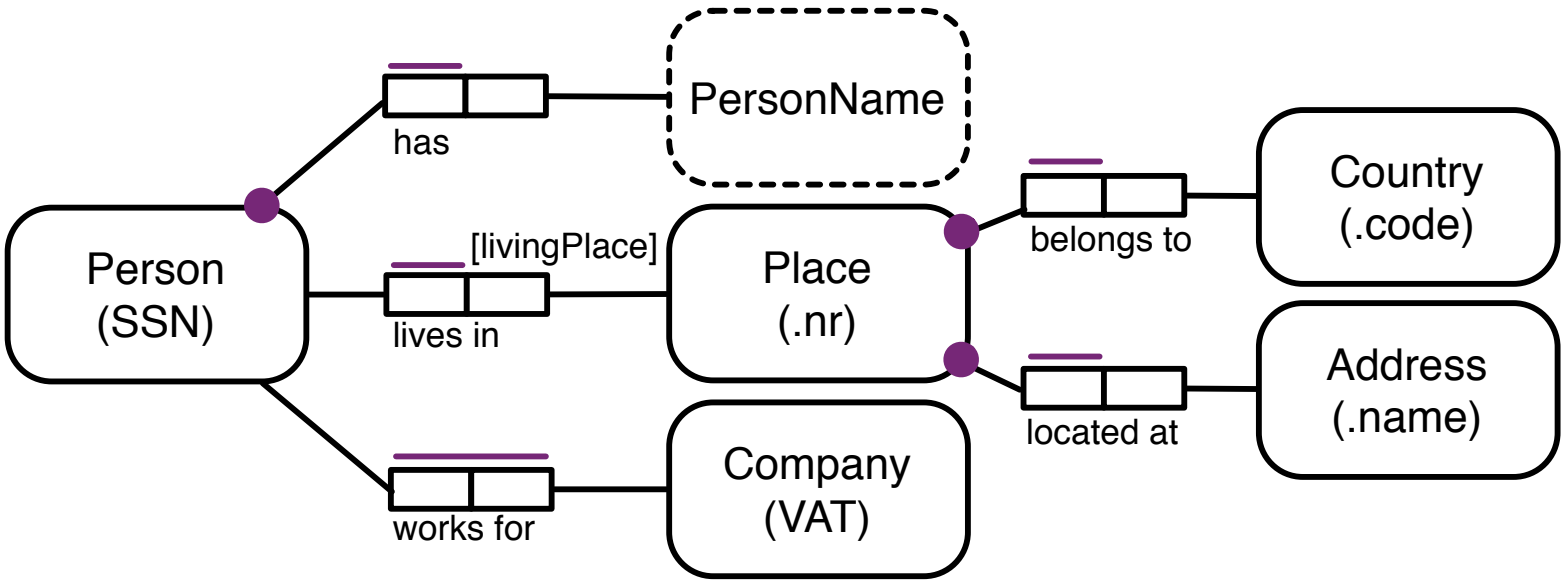
Case 2

- Applies to binary fact types with simple UCs all over roles played by the same object type.
- Mandatory dots determine mandatory columns in the grouped table.

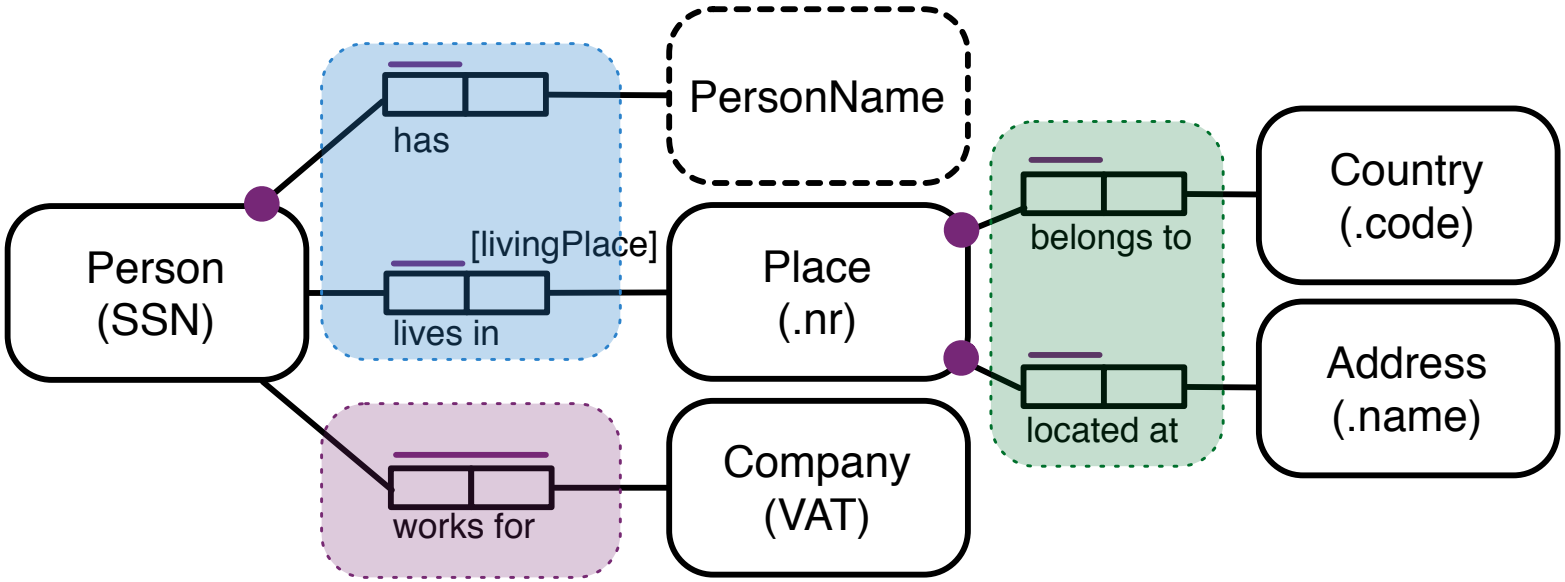
Case 1

- Applies to binary fact types with spanning UCs and to n-ary fact types with n $>$ 2 (why?).
- FKs must be added for those roles attached to object types that play mandatory roles elsewhere ($\subseteq$).

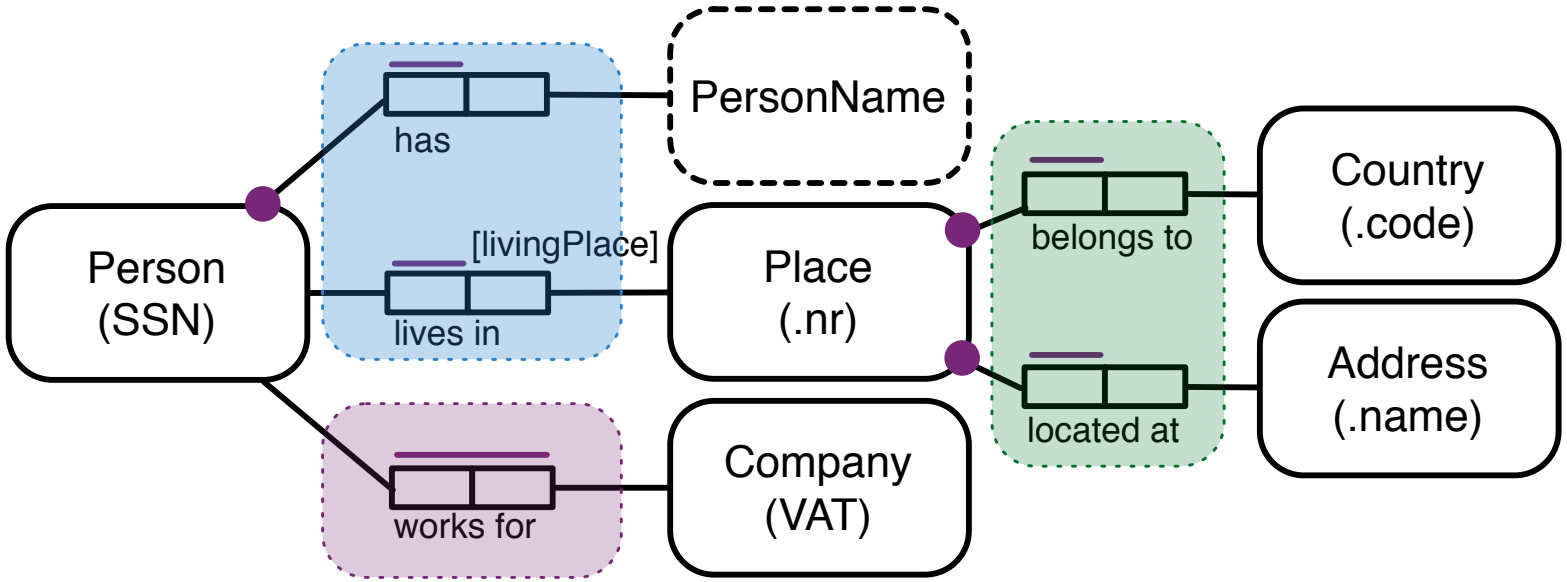
Example


PersonName
has
Person
(SSN)
[livingPlace]
lives in
Place
(.nr)
belongs to
Country
(.code)
Address
(.name)
located at
Company
(VAT)
works for

Example


PersonName
has
Person
(SSN)
[livingPlace]
lives in
Place
(.nr)
belongs to
Country
(.code)
Address
(.name)
located at
Company
(VAT)
works for

Example



Place(placeNr, countryCode, addrName)

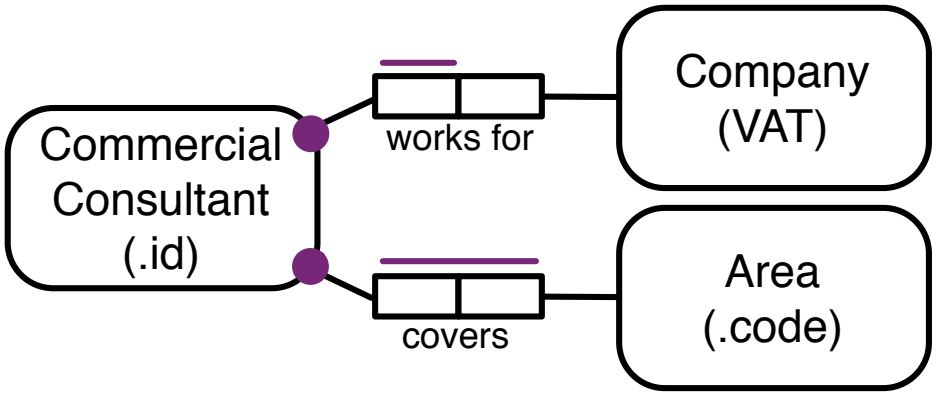
Person(SSN, persName, [livingPlace])

WorksFor(SSN, VAT)

N.B.: squared brackets in ORM schema denote role name, while in relational schema they denote optionality.

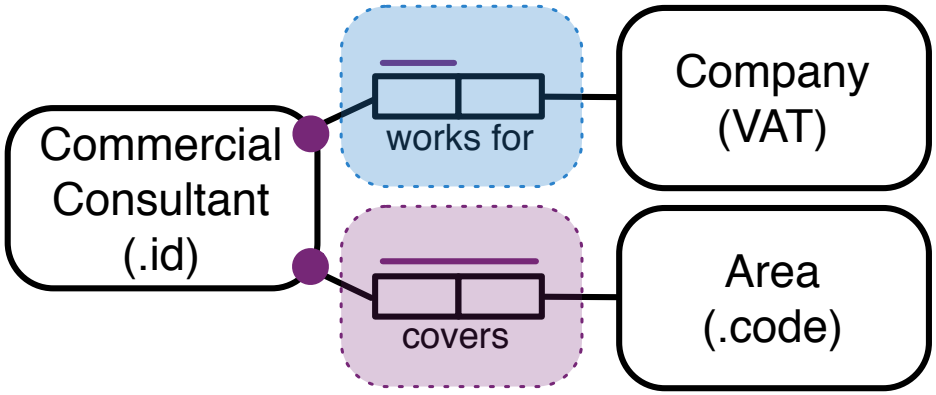
Example - Referential Cycle

The two mandatory roles form an equality constraint.



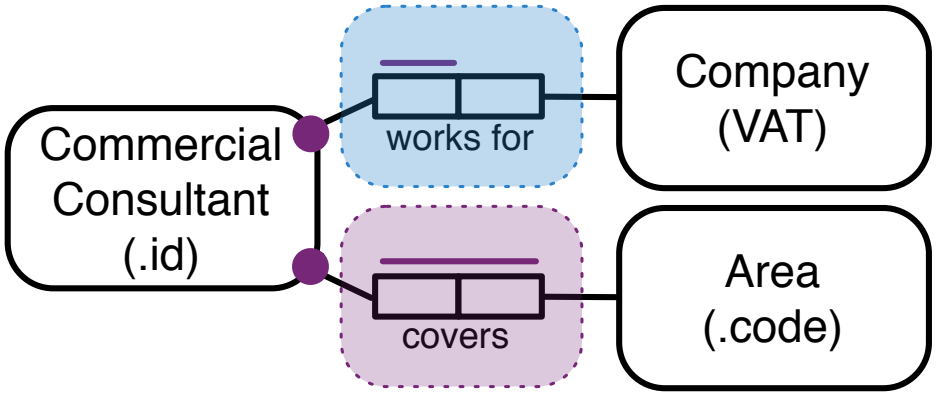
Example - Referential Cycle

The two mandatory roles form an equality constraint.



Example - Referential Cycle

The two mandatory roles form an equality constraint.



CommercialConsultant(commConsId, VAT)

Covers(commConsId, AreaCode)

The equality constraint becomes a pair of subset constraints in the relational schema, forming a referential cycle.

- The subset constraint from `Covers` to `CommercialConsultant` is a FK constraint.
- The subset constraint from `CommercialConsultant` to `Covers` is **not** a FK constraint, because it targets only part of the PK in `Covers`. It can be enforced using assertions, stored procedures, triggers, ...

Referential cycles should be avoided if possible.

Mapping 1:1 Associations

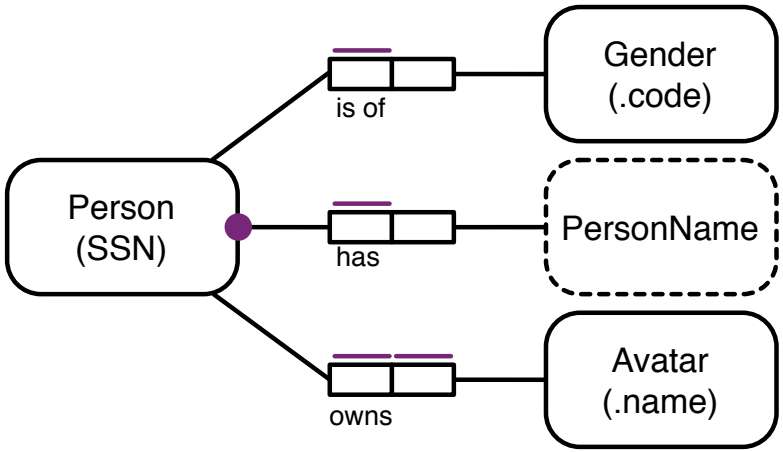
The key point is on which side the association must be grouped. Aspects to be considered:

- Presence of null values must be minimized.
- Do both object types play other functional roles?
- Are the involved roles mandatory on both sides?

1:1 Association - Only One "Complex" Endpoint

Hypothesis

We are **sure** that one of the two endpoints *does not* play other functional roles.



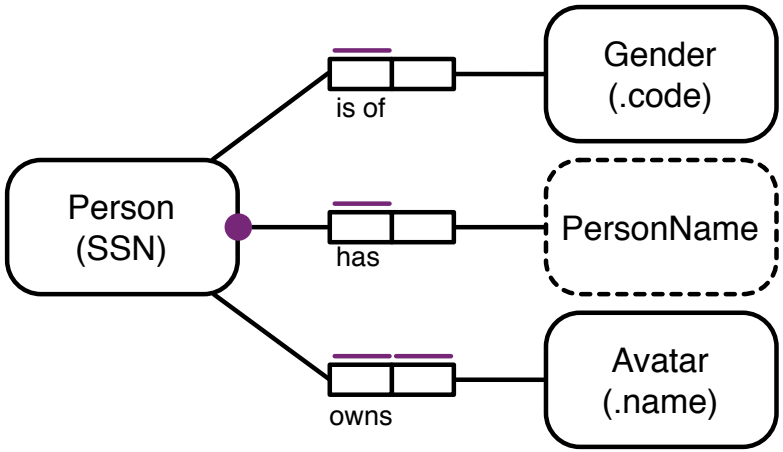
Strategy

Group around the other endpoint.

1:1 Association - Only One "Complex" Endpoint

Hypothesis

We are **sure** that one of the two endpoints *does not* play other functional roles.



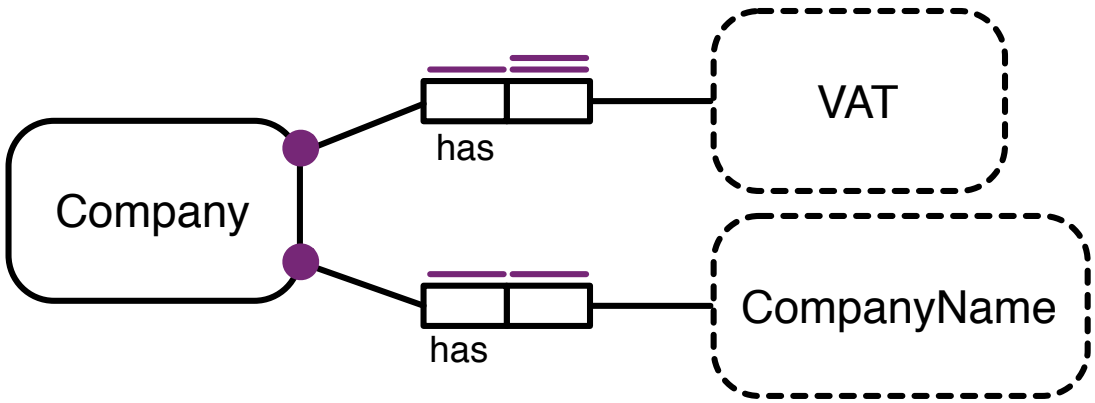
Person(SSN, [gender], persName, [avName])

Strategy

Group around the other endpoint.

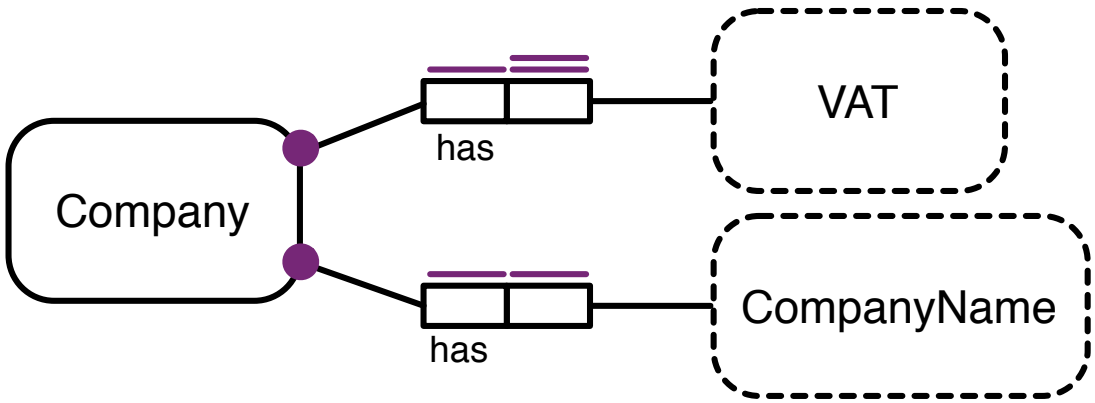
1:1 Association - Only One "Complex" Endpoint

This case directly covers 1:1 associations with value types $\rightarrow$ grouping on the object type.



1:1 Association - Only One "Complex" Endpoint

This case directly covers 1:1 associations with value types → grouping on the object type.

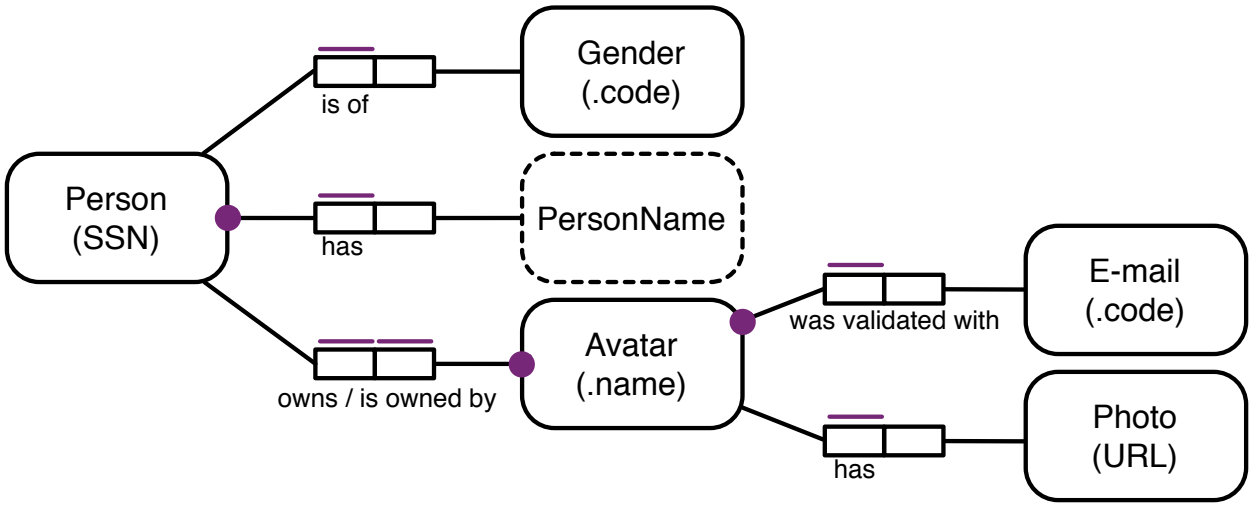


Company(VAT, companyName)

1:1 Association - Only One Mandatory Endpoint

Hypothesis

Both endpoints play other functional roles, but only one endpoint role is mandatory.



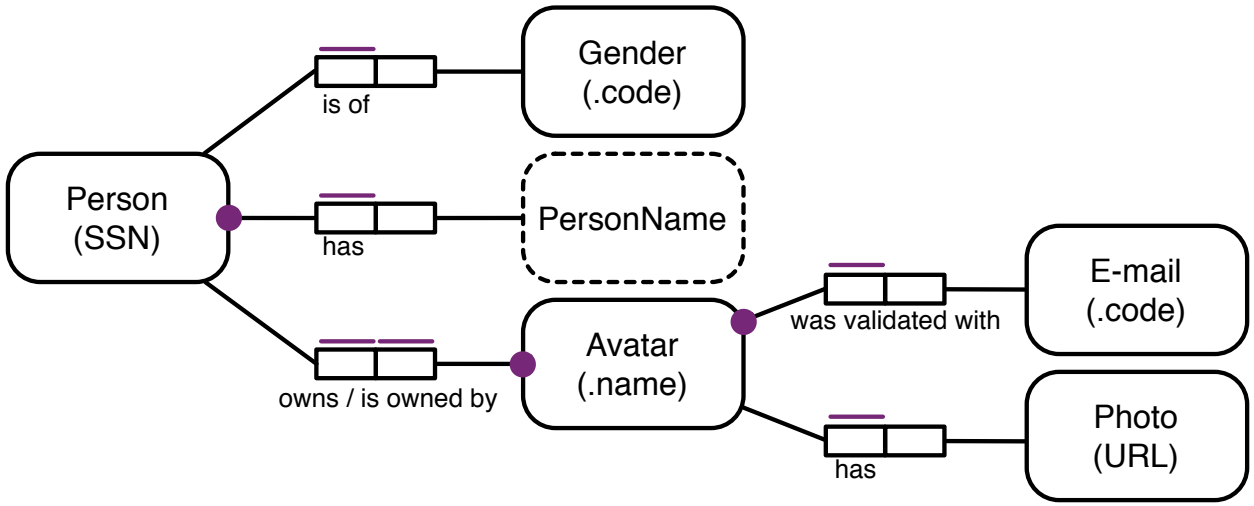
Strategy

Group on the mandatory role side.

1:1 Association - Only One Mandatory Endpoint

Hypothesis

Both endpoints play other functional roles, but only one endpoint role is mandatory.



Person(SSN, [gender], persName)

Avatar(avName, SSN, eMailCode, [photoURL])

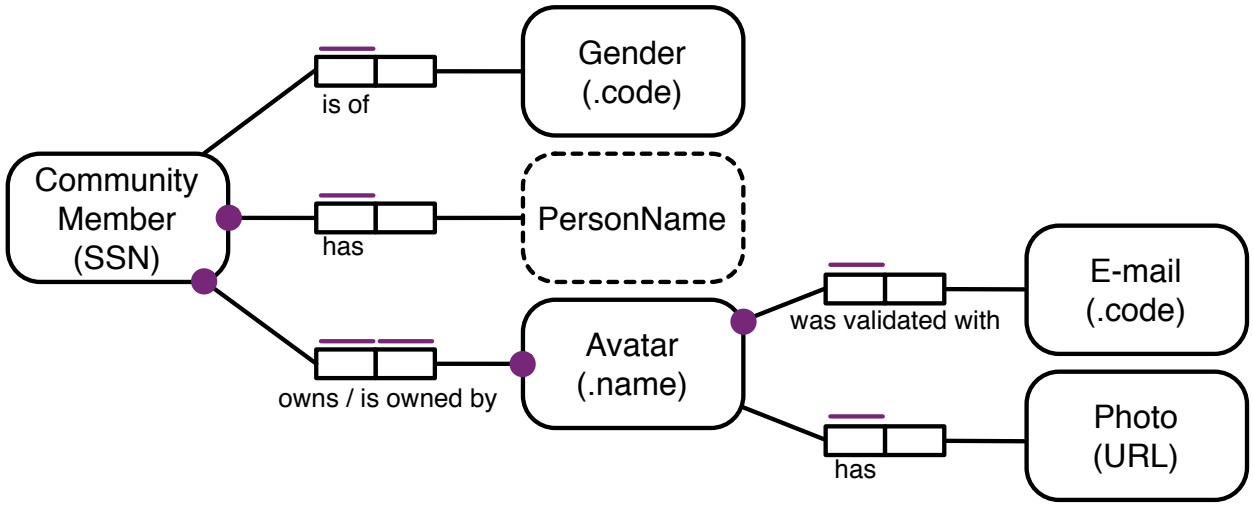
Strategy

Group on the mandatory role side.

1:1 Association - Symmetric Case with Both Mandatory

Hypothesis

Both endpoints play other functional roles, and both endpoint roles are mandatory.



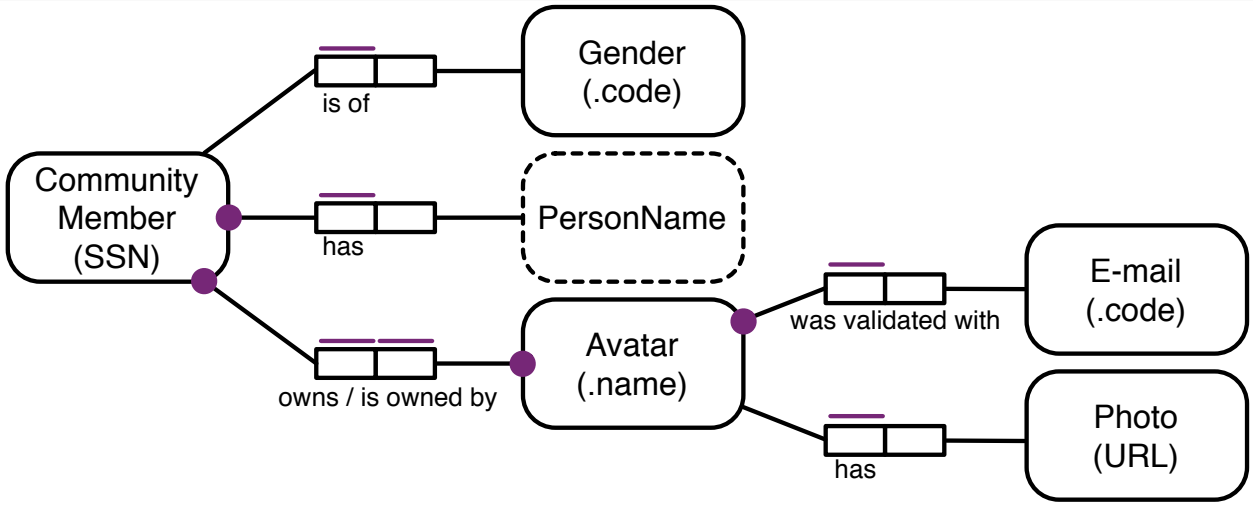
Strategy

Arbitrarily choose one of the two endpoints (free choice of the modeler).

1:1 Association - Symmetric Case with Both Mandatory

Hypothesis

Both endpoints play other functional roles, and both endpoint roles are mandatory.



(1) CommunityMember(SSN, [gender], persName)

Avatar(avName, SSN, eMailCode, [photoURL])

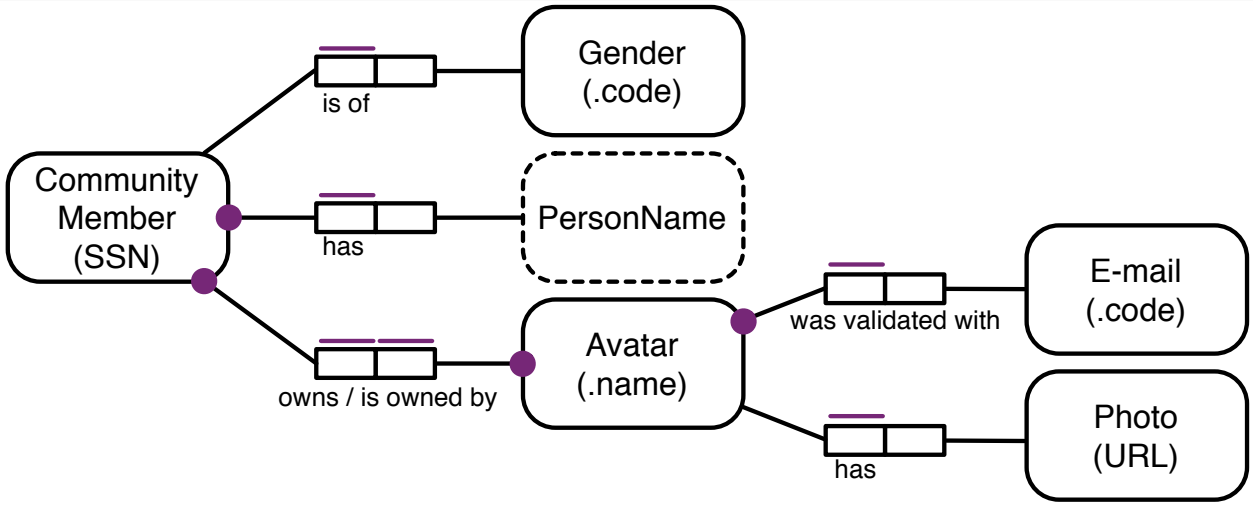
Strategy

Arbitrarily choose one of the two endpoints (free choice of the modeler).

1:1 Association - Symmetric Case with Both Mandatory

Hypothesis

Both endpoints play other functional roles, and both endpoint roles are mandatory.



(2) CommunityMember(SSN, avName, [gender], persName)

Avatar(avName, eMailCode, [photoURL])

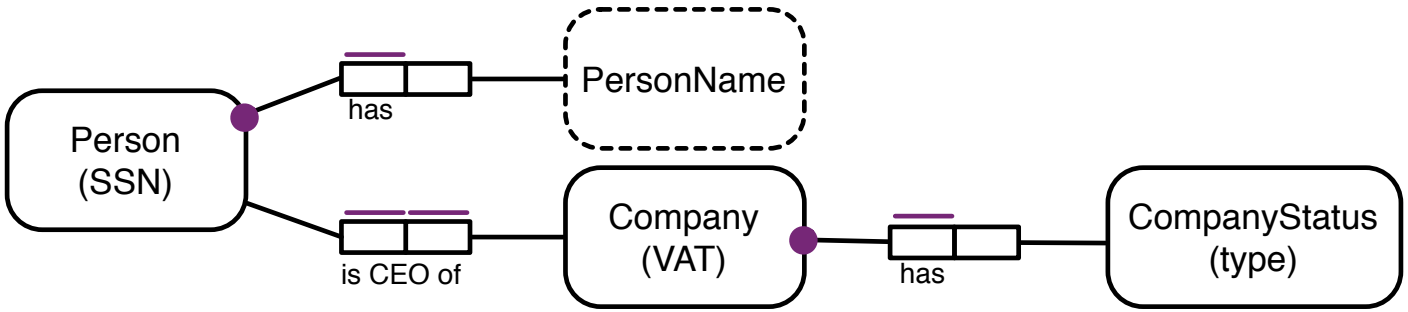
Strategy

Arbitrarily choose one of the two endpoints (free choice of the modeler).

1:1 Association - Symmetric Case with None Mandatory

Hypothesis

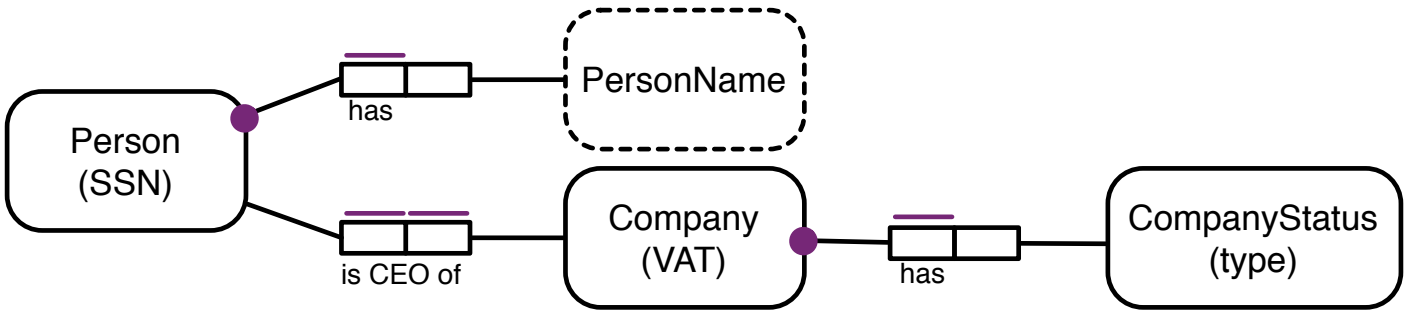
Both endpoints play other functional roles, and none of the endpoint roles is mandatory.



Strategy

Either grouping is reasonable.
Choose the best grouping based on the percentage of likely null values.
If both solutions are unsatisfactory, introduce a third table for the 1:1 association.

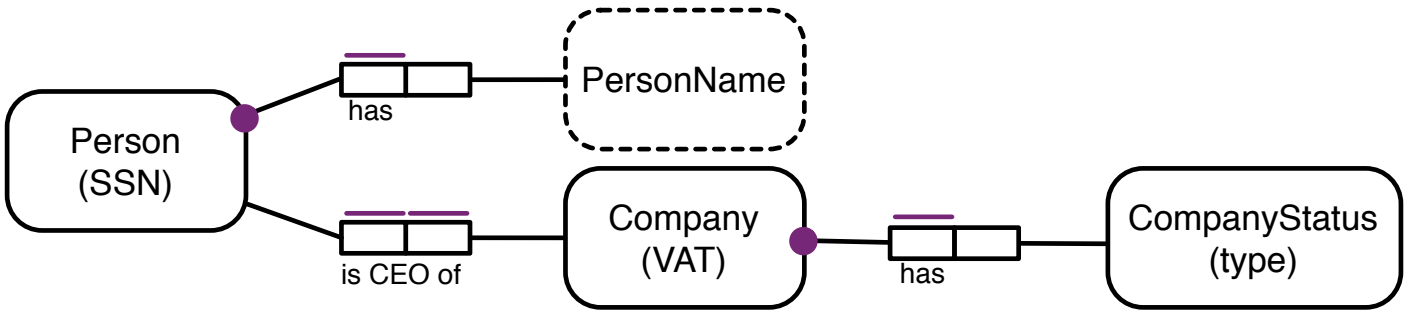
1:1 Association - Symmetric Case with None Mandatory



Hp: it is more likely for a Company to have a CEO that for a Person to be CEO of a Company.

Hp: it is likely that both solutions (grouping on Person or on Company) yield many null values.

1:1 Association - Symmetric Case with None Mandatory



Hp: it is more likely for a Company to have a CEO that for a Person to be CEO of a Company.

Person(SSN , persName)

Company(VAT, [SSN], ComStatus)

Hp: it is likely that both solutions (grouping on Person or on Company) yield many null values.

Person(SSN, persName)

IsCEOof(SSN, VAT)

Company(VAT, ComStatus)

Mapping 1:1 Associations - Overall Strategy

1. **If** only one object type in the association has another functional rule
   then group on its side.
2. **Else if** both object types have other functional roles and only one role in the 1:1 association is mandatory
   then group on its side.
3. **Else if** no object type has another functional role
   then map 1:1 to a separate table.
4. **Else** the grouping choice is completely delegated to the modeler.

Mapping External UCs

Procedure:

1. Do not consider the identification scheme of object types.
2. Group fact types into tables, using compact surrogates as columns.
3. Restore the full tables replacing surrogates with the attributes used for preferred identification.

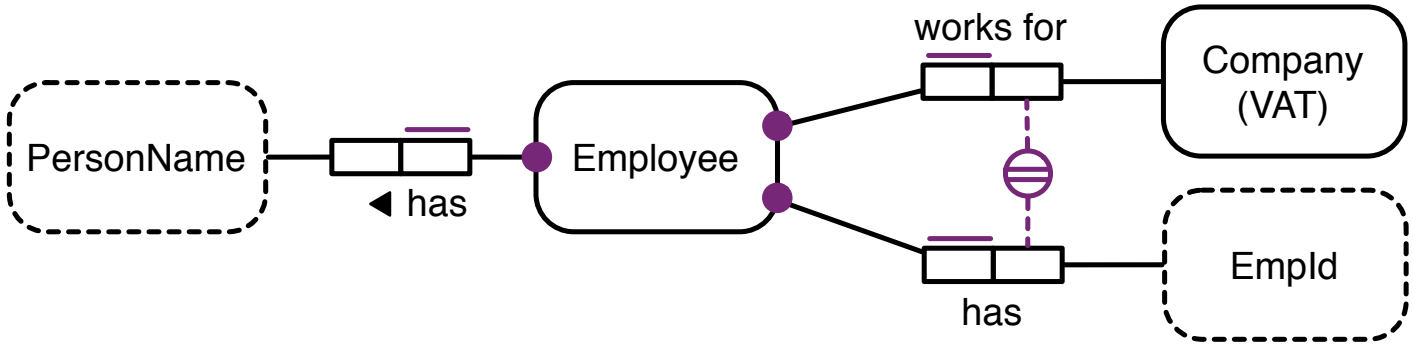
Cases:

- External UC is the preferred identification scheme for an object type attached to other functional roles.
- External UC is the preferred identification scheme for an object type attached to other nonfunctional roles.
- External UC is not the preferred identification scheme for an object type attached to other functional roles.
- External UC is not the preferred identification scheme for an object type, and it is paired with an n:m fact type.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



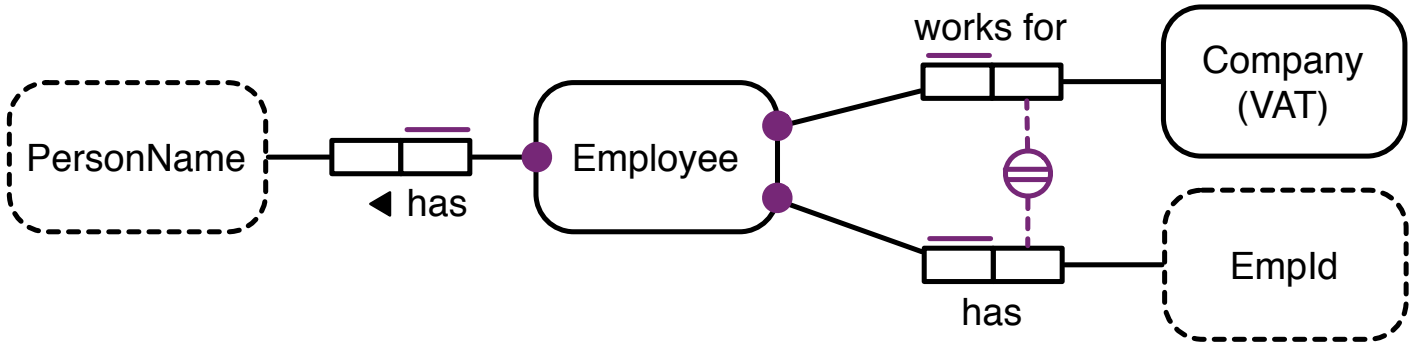
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



Employee(e, n)

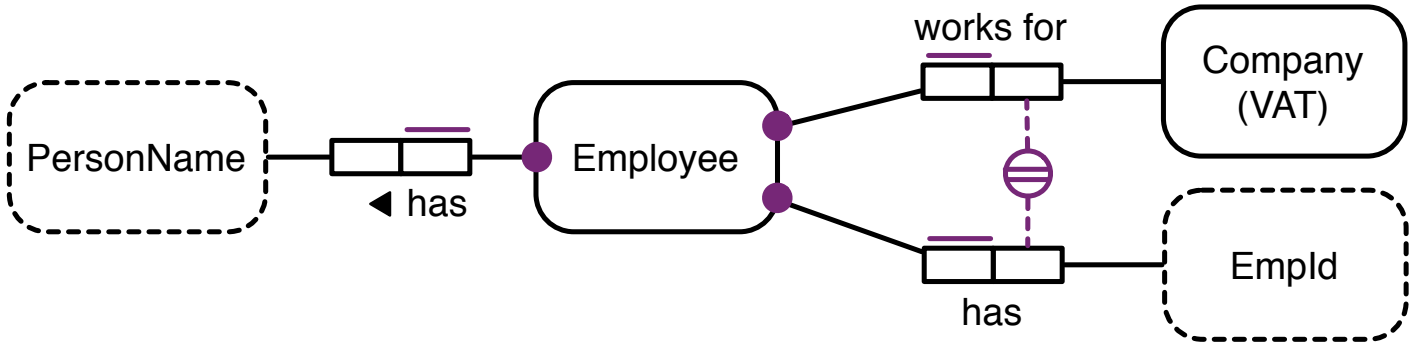
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



Employee(empId, VAT, empName)

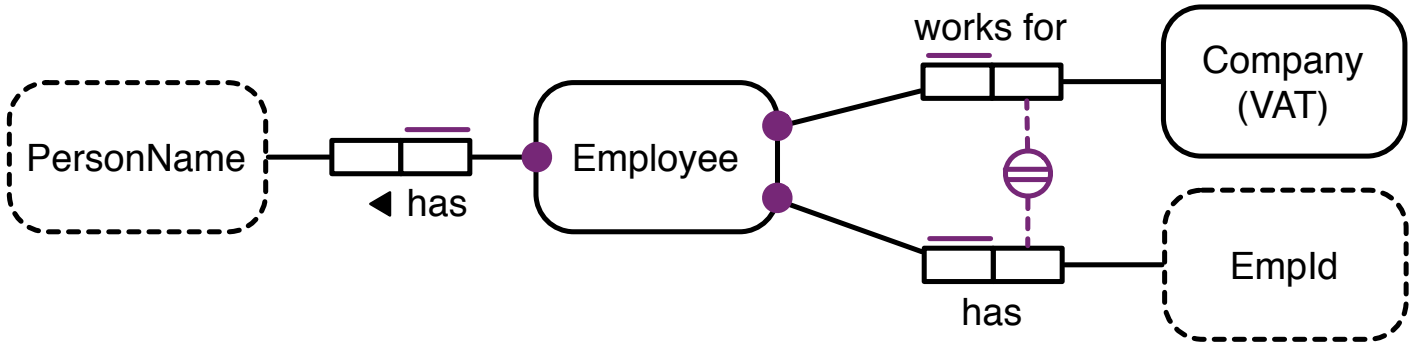
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



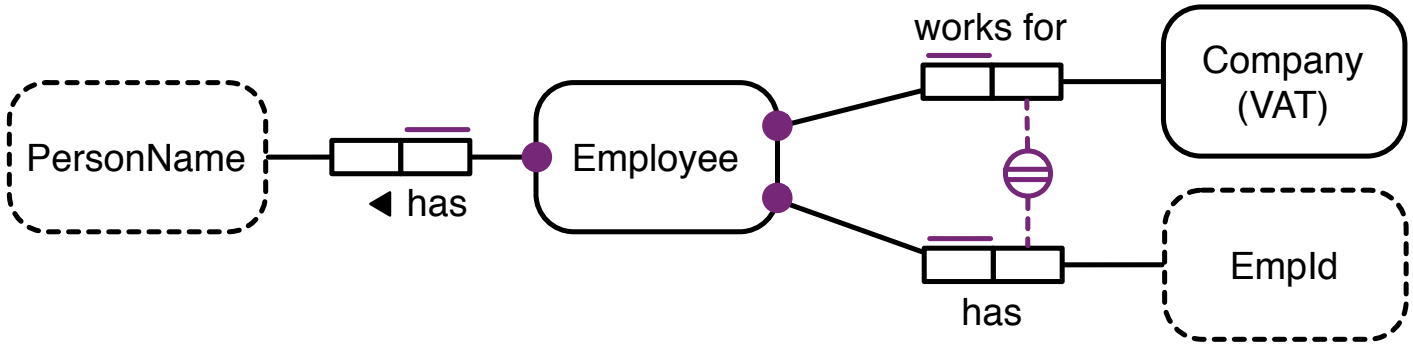
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



Employee(e, n)

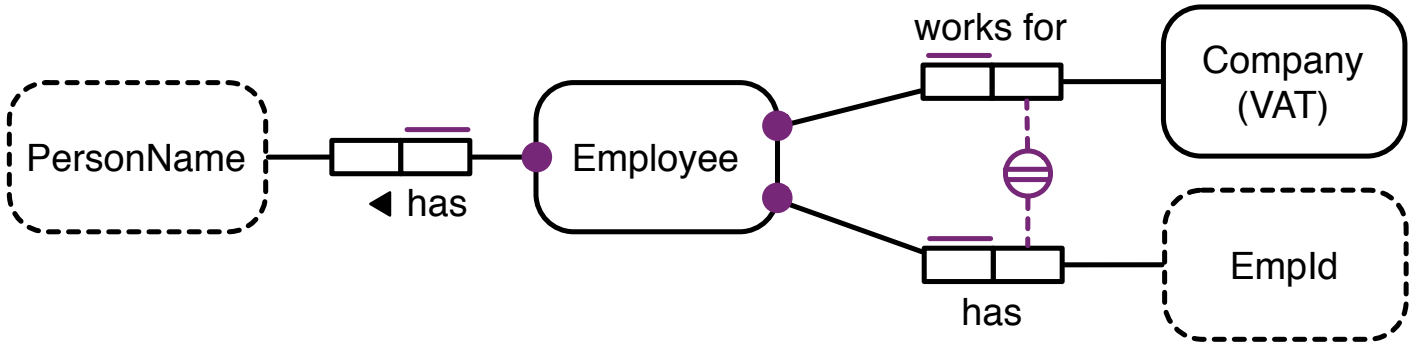
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Functional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and plays other functional roles.



Employee(empId, VAT, empName)

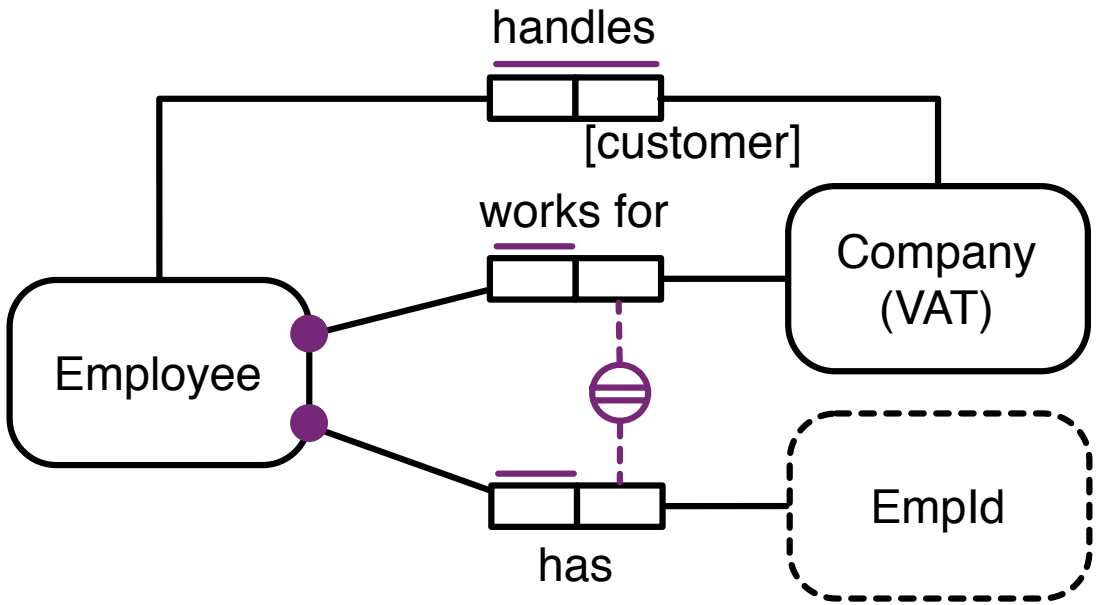
Strategy

Group on the object type using surrogates, without considering the predicates involved in the external UC.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Nonfunctional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and has an m:n association with another object type



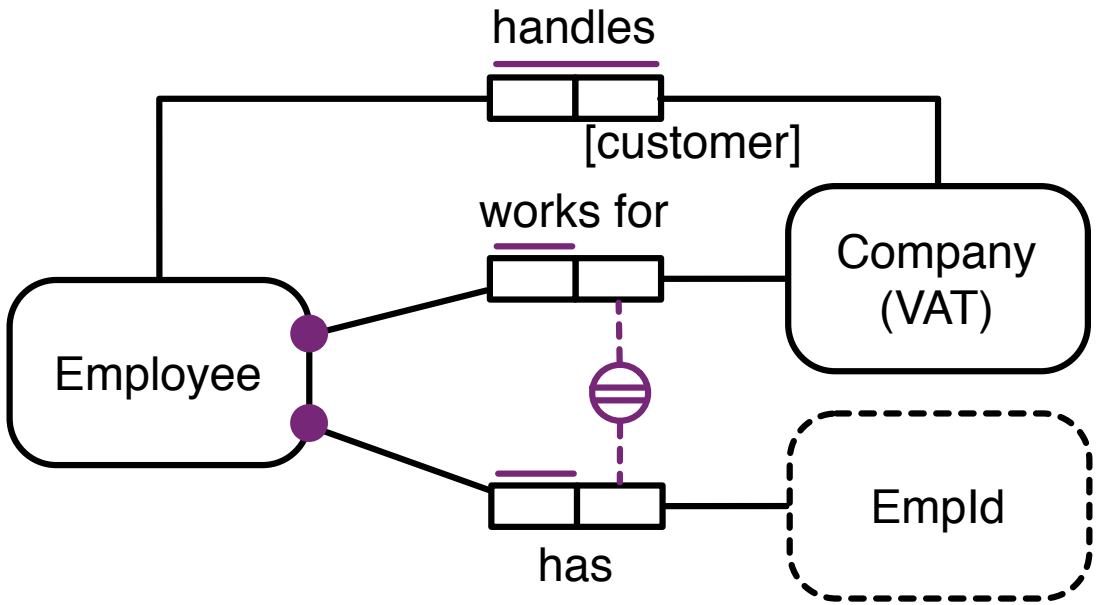
Strategy

Map the m:n association to a separate table, using surrogates.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Nonfunctional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and has an m:n association with another object type



Handles(e, c)

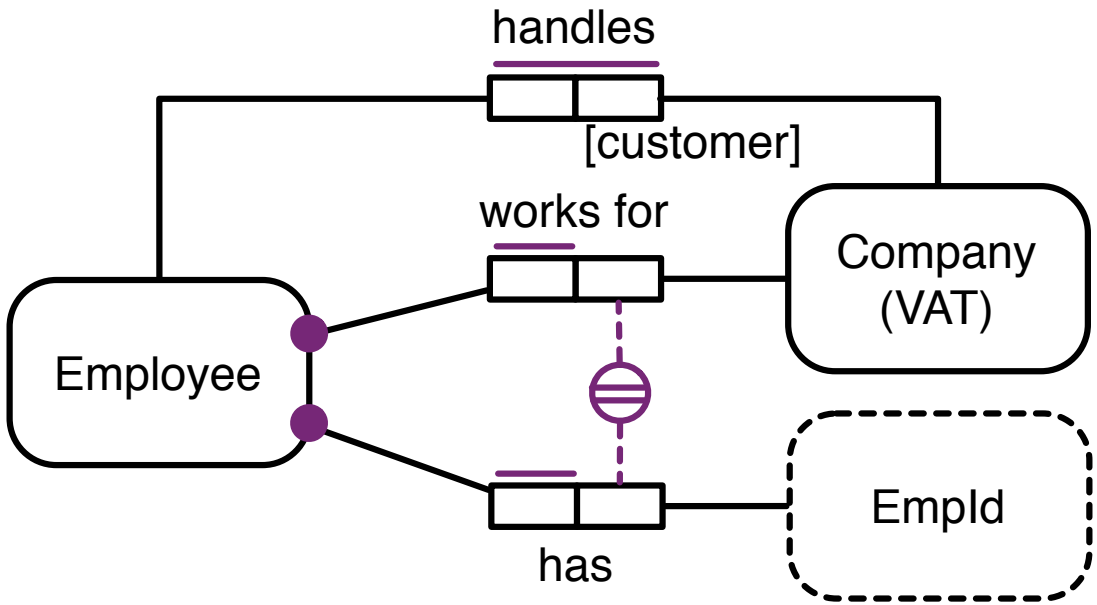
Strategy

Map the m:n association to a separate table, using surrogates.
Then expand the PK using the object types involved in the external UC.

External Preferred UC, Nonfunctional Fact Type

Hypothesis

Object type has an external UC as preferred identifier, and has an m:n association with another object type



Handles(empId, VAT, customerVAT)

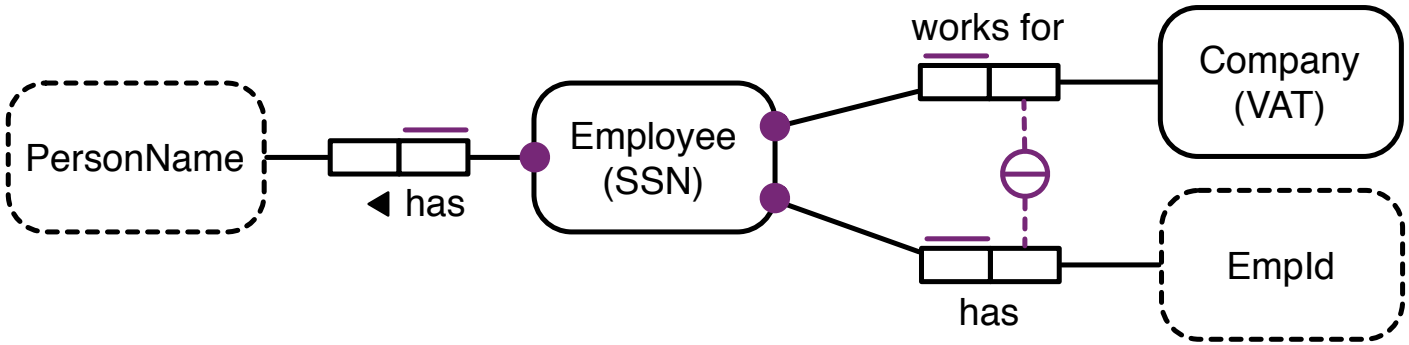
Strategy

Map the m:n association to a separate table, using surrogates.
Then expand the PK using the object types involved in the external UC.

External UC, Functional Fact Type

Hypothesis

Object type with a simple preferred identifier, attached to functional roles with an external UC and other functional roles.



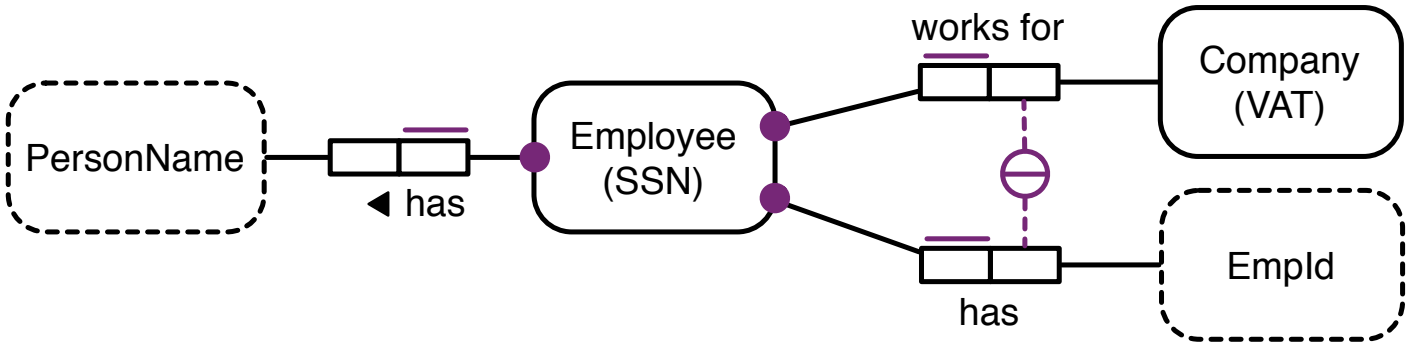
Strategy

Follow the standard mapping. Model the external UC as a key.

External UC, Functional Fact Type

Hypothesis

Object type with a simple preferred identifier, attached to functional roles with an external UC and other functional roles.



Employee(SSN, VAT, empId, persName)

Strategy

Follow the standard mapping. Model the external UC as a key.

External UC Involving m:n Fact Type

Hypothesis

External UC involving an m:n fact type.

Strategy

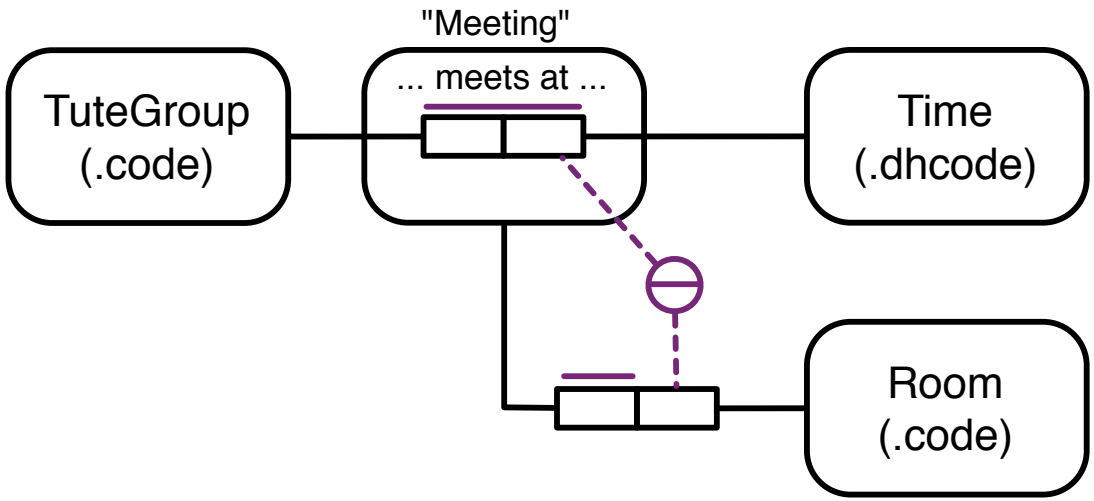
Follow the standard mapping without considering the external UC. Add the external UC as an inter-table constraint.

Mapping Objectified Associations

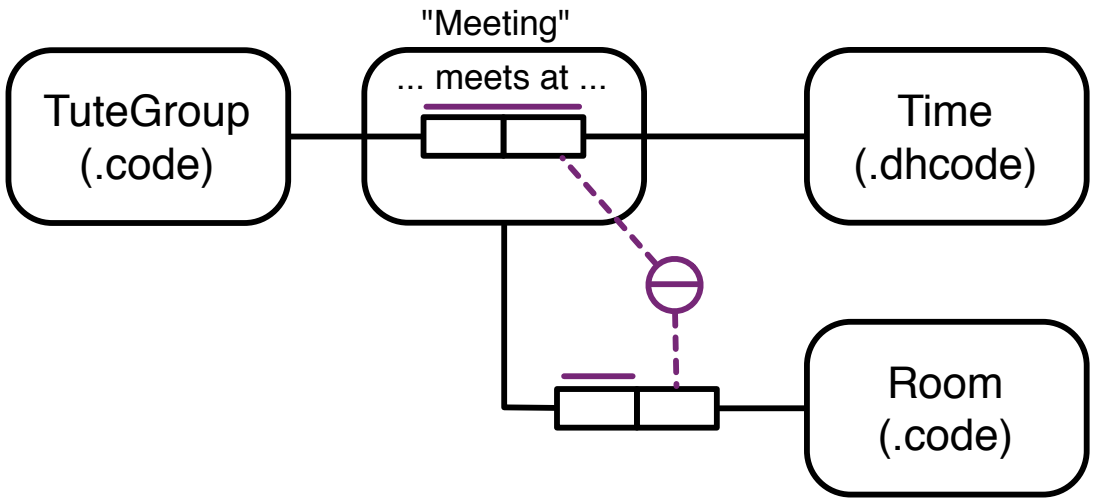
Procedure:

1. Do not consider the identification scheme of the objectified association.
2. Consider the objectified association as a black box.
3. Group fact types in the standard way.
4. Unpack the black box into its component attributes.
5. Deal with fine-grained constraints involving component roles of the objectified association.

Example

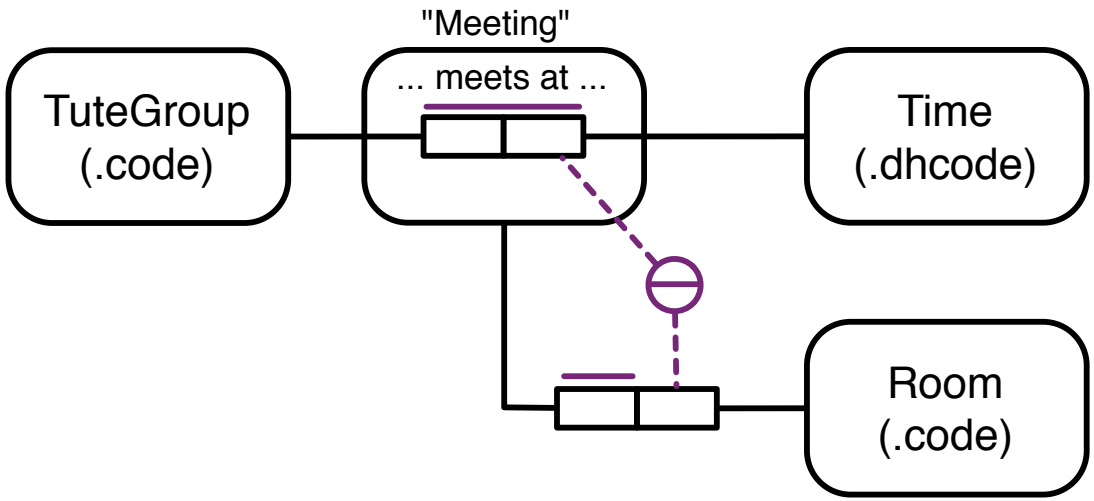


Example



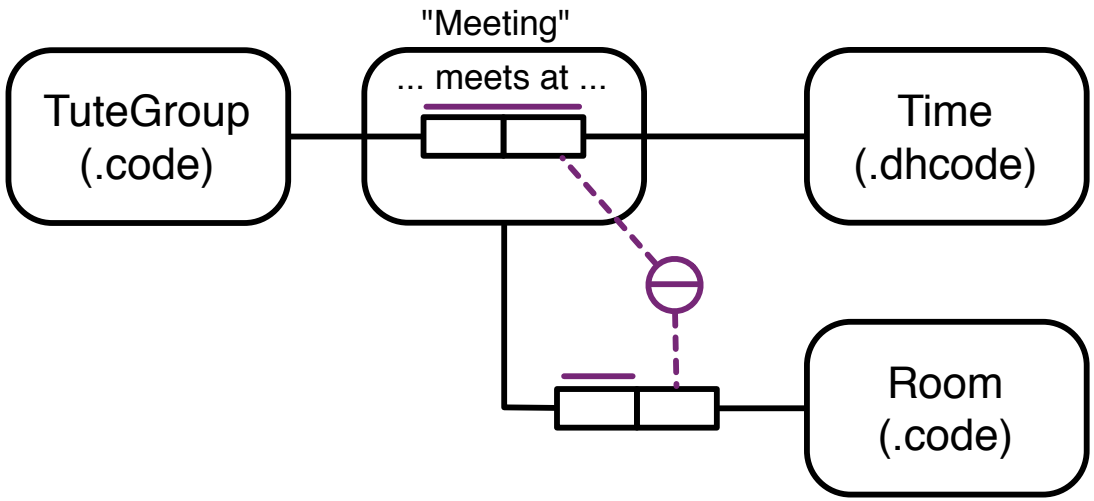
- Consider the objectified association as a black box: Meeting (<u>■</u>, [roomCode])

Example



- Consider the objectified association as a black box:
  Meeting (<u>■</u>, [roomCode])
- Expand:
  Meeting (<u>tuteCode, meetingTime</u>, [roomCode])

Example

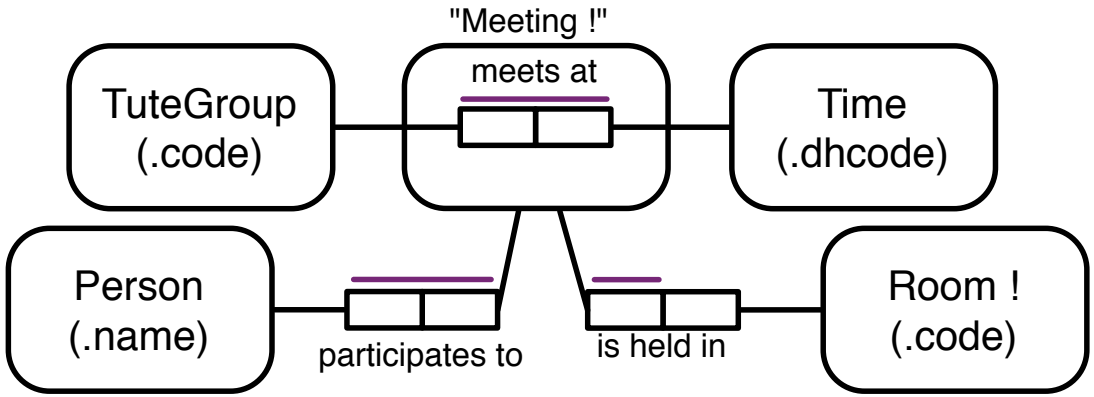


- Consider the objectified association as a black box:
  Meeting (■, [roomCode])
- Expand:
  Meeting (tuteCode, meetingTime, [roomCode])
- Incorporate fine-grained constraints: key meetingTime, [roomCode]

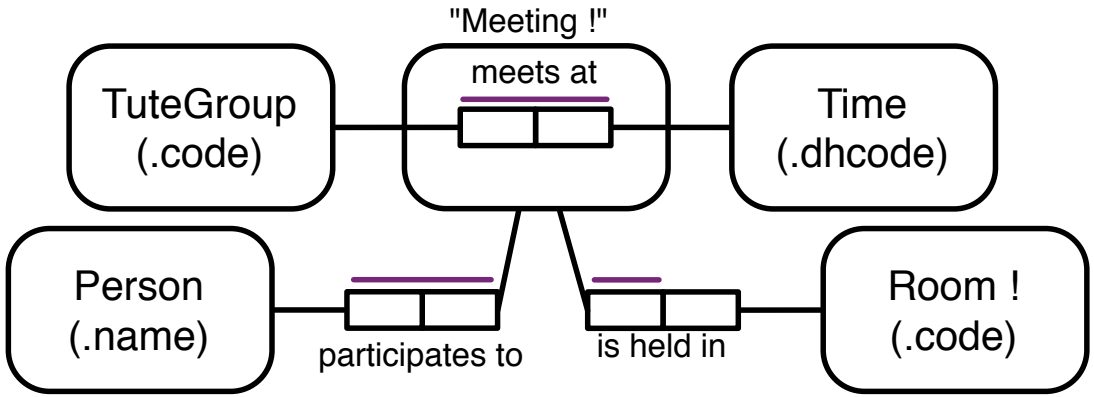
Mapping Independent Object Types

Each independent object type has its own life cycle, hence:

- it must be mapped to a dedicated table with its preferred identifier as PK, together with all fact types in which it plays a functional role; (note: they are all optional by construction, can you spot why?)
- every nonfunctional role will have a FK pointing to this table.



Example

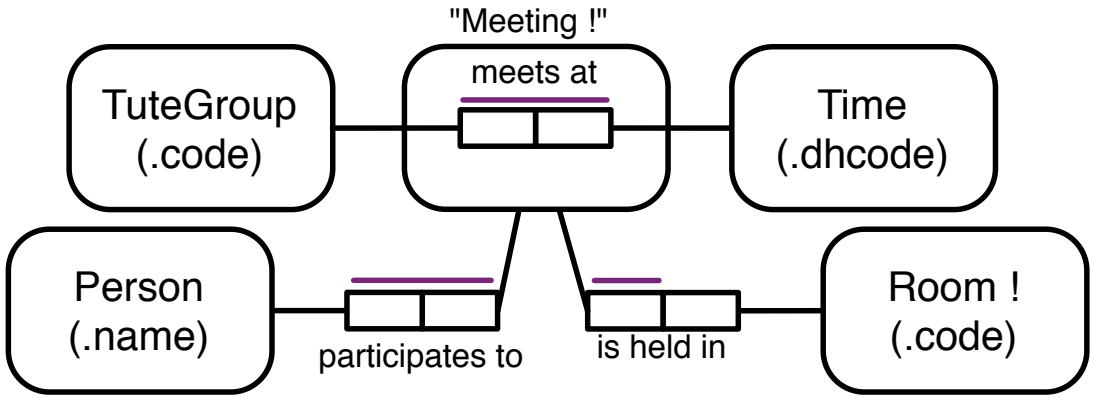


Room(roomCode)

Meeting(■, [roomCode])

PartTo(persName, ■)

Example

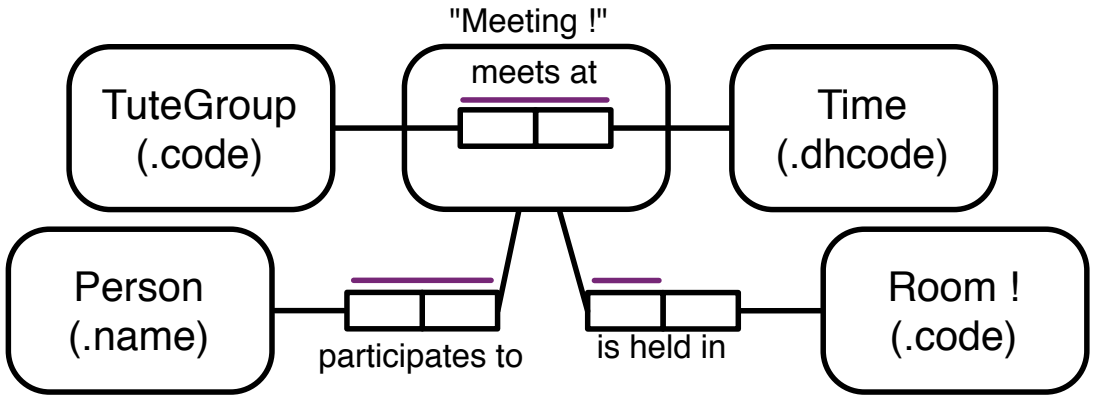


Room(roomCode)

Meeting(■, [roomCode])

PartTo(persName, ■)

Example



Room(roomCode)

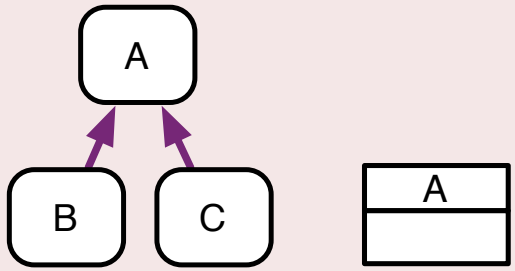
Meeting(tGroup, tTime, [roomCode])

PartTo(persName, tGroup, tTime)

Mapping Subtype Constraints

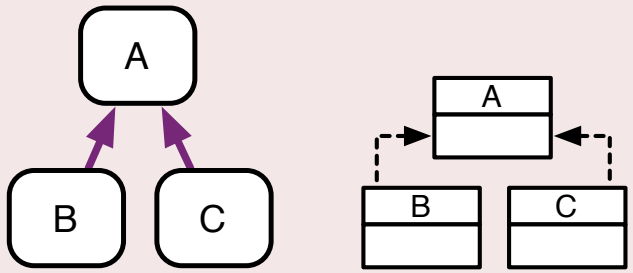
Absorption

Subtypes are absorbed back into their top supertype, grouping the fact types as usual and adding the subtyping constraints as textual qualifications.



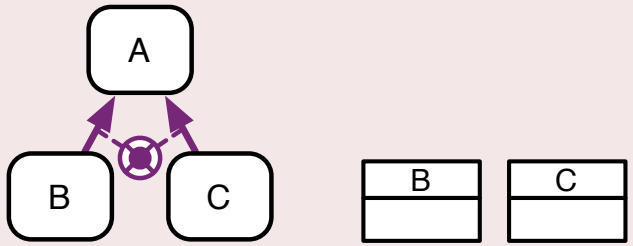
Separation

Each object type of the hierarchy is mapped to a separate table. FKs are added from the subtypes tables to the supertype table.



Partition

Supertype is removed, replicating the attached information for each subtype.

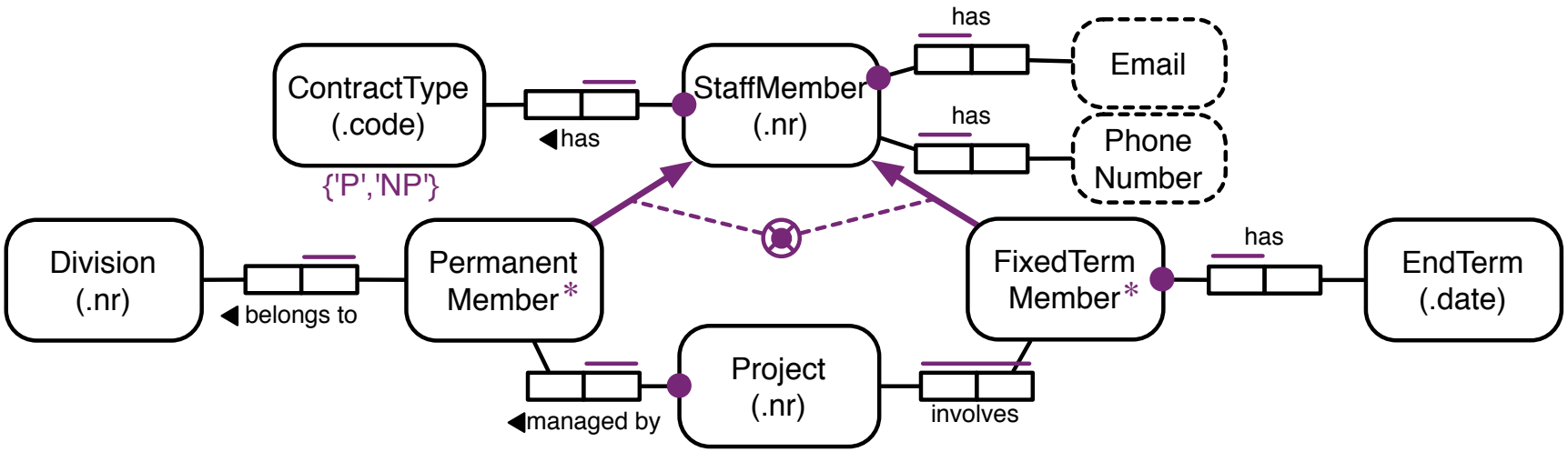


Absorption

Standard approach for mapping hierarchies.

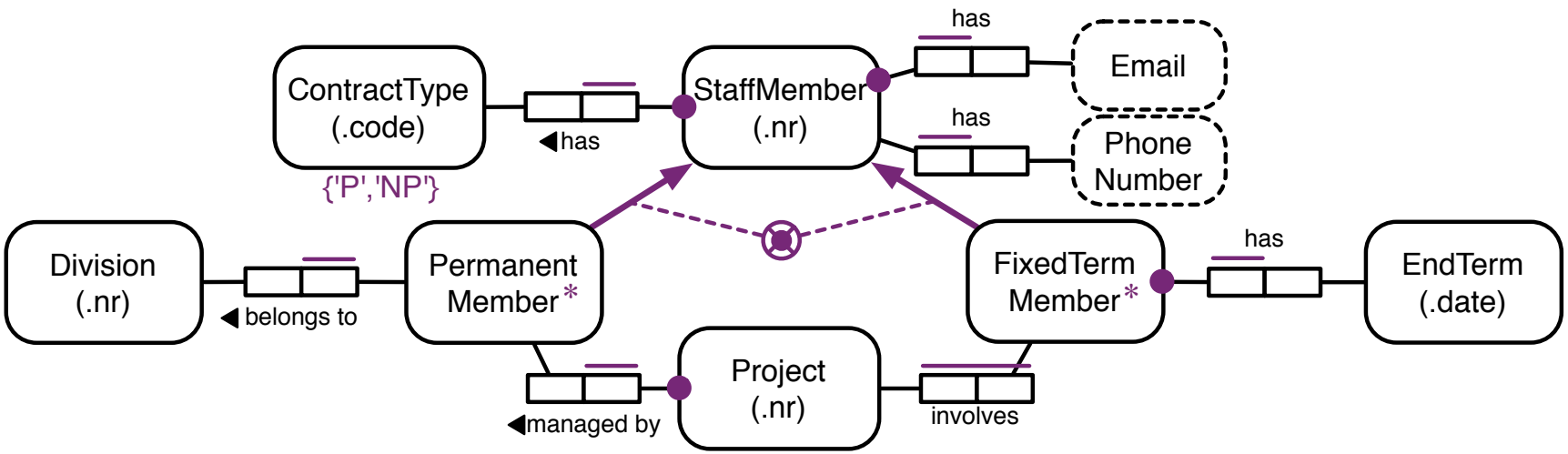
- A *discriminator* column typically reflects the presence of a classification type to distinguish subtypes (e.g., Gender, CompanyStatus, MemberType) with a fixed domain whose possible values correspond to the different subtypes.
- All fact types attached to subtypes are moved to the supertype, making the participation of the supertype *optional*.
- Textual constraints are added to specify when such a participation is in fact mandatory, using the discriminator column.
  - **`exists only if`** and **`exists iff`** constraints.
- M:n fact types involving subtypes are mapped to separate tables, with FK pointing to the supertype table, and suitably combined with constraints that use the discriminator column.
  - **`only where`** constraints.
- Main advantage: only one table for the hierarchy (no join, greater efficiency).
- Main weakness: many potential null values, difficult to pose queries regarding only subtypes.

Absorption - Representative Example



*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'

Absorption - Representative Example



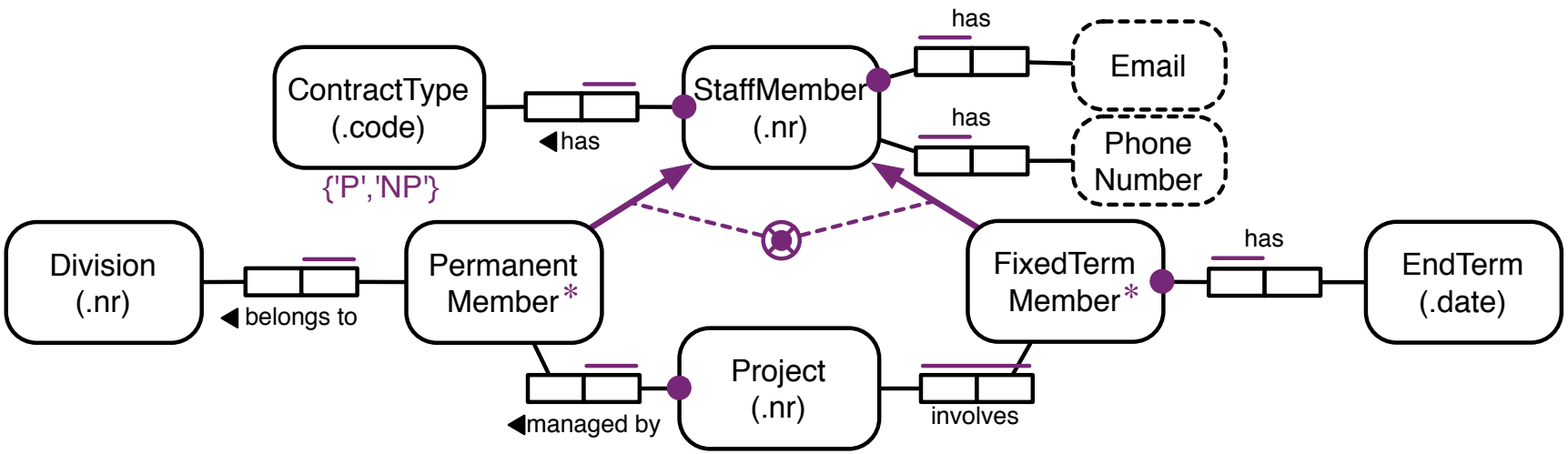
*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'

Project(projNr, manager)

ProjectInvolvesFTMember(projNr, ftMember)

StaffMember(staffNr, contrCode, eMail, [phoneNumber], [divisionNr] , [endTerm])
{'P','NP'}

Absorption - Representative Example



*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'

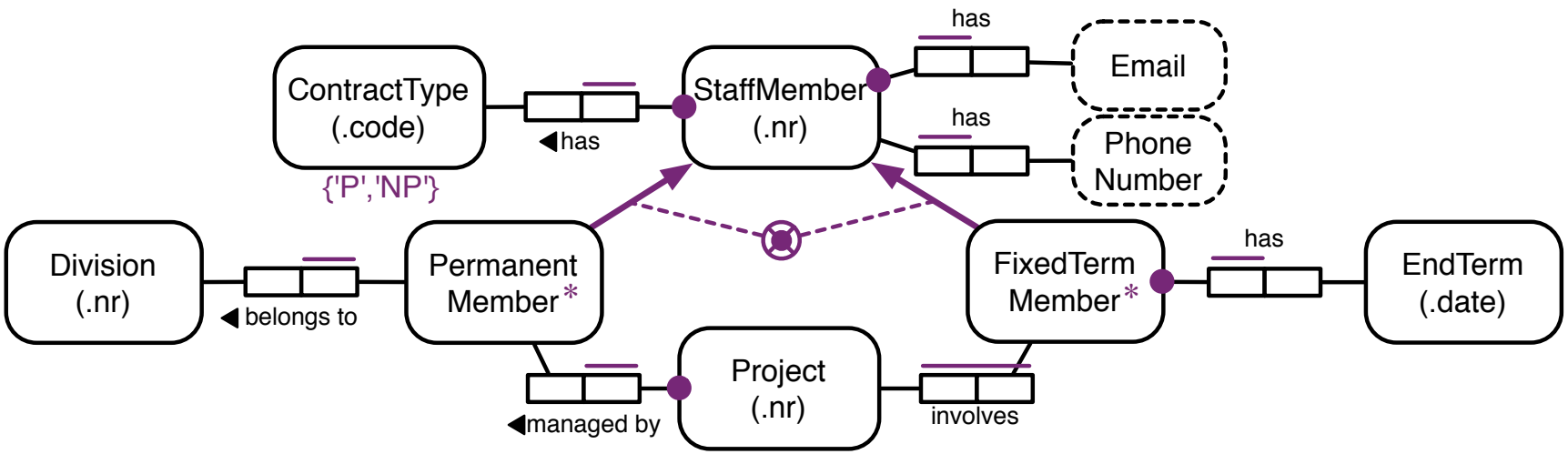
Project(projNr, manager)

ProjectInvolvesFTMember(projNr, ftMember)

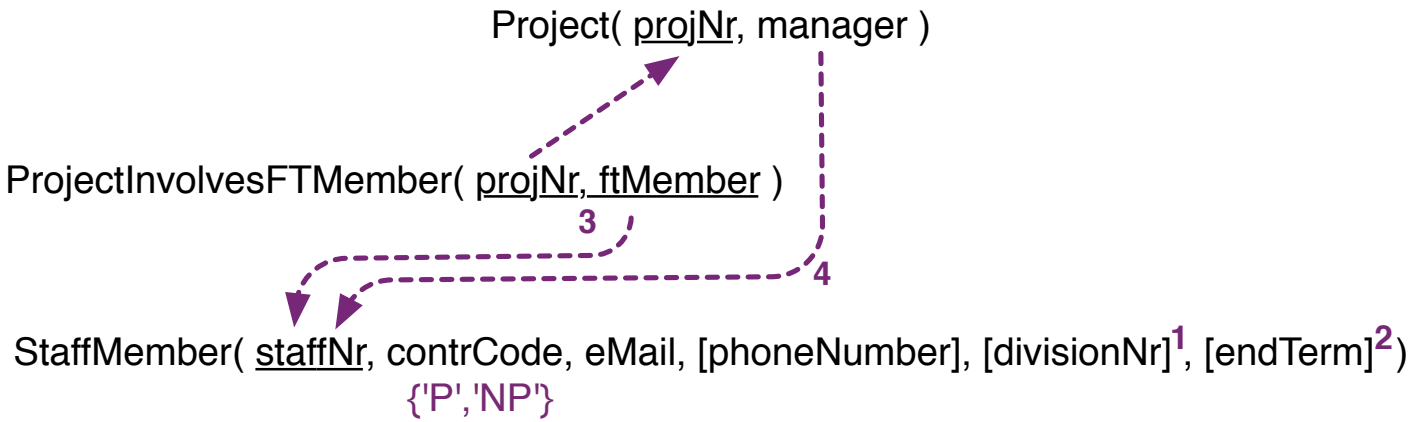
StaffMember(staffNr, contrCode, eMail, [phoneNumber], [divisionNr][1], [endTerm][2])
{'P','NP'}

[1]`exists only if contrCode = 'P'` [2]`exists iff contrCode = 'NP'`

Absorption - Representative Example



*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'



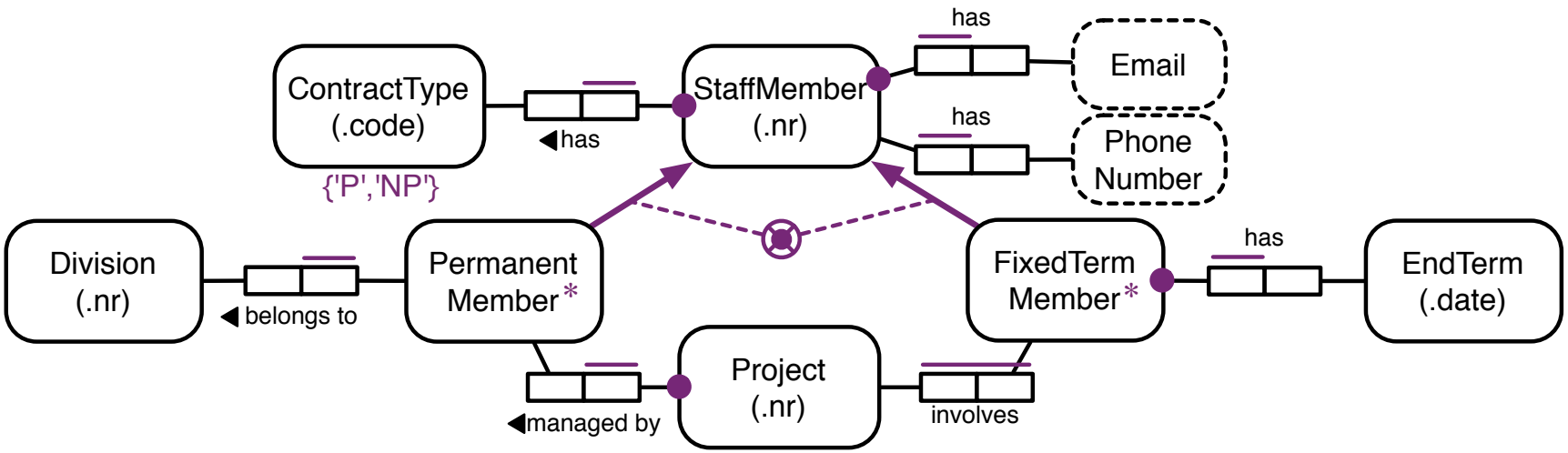
[1]**exists only if** contrCode = 'P' [2]**exists iff** contrCode = 'NP'
[3]**only where** contrCode = 'NP' [4]**only where** contrCode = 'P'

Separation

Mapping that tends to maintain the structure of the hierarchy, representing supertype and subtypes separately.

- Each object type of the hierarchy becomes a separate table, provided that it is involved in at least one functional role.
- PK of each subtype refers to (FK) the PK of the supertype (subtype constraints become subset constraints).
- Also in this case, a discriminator column can be typically used in the supertype; this requires the presence of suitable constraints in the FKs.
  - Depending on the existence of a mandatory role in the supertype, the constraint has the shape of **`exactly where`** or **`only where`**.
- Full information about the subtype is obtained by natural join with the table of the supertype.
- Main advantage: null values minimization, easy to access subtypes.
- Main weakness: joins needed, slow insertions for subtypes.

Separation - Representative Example



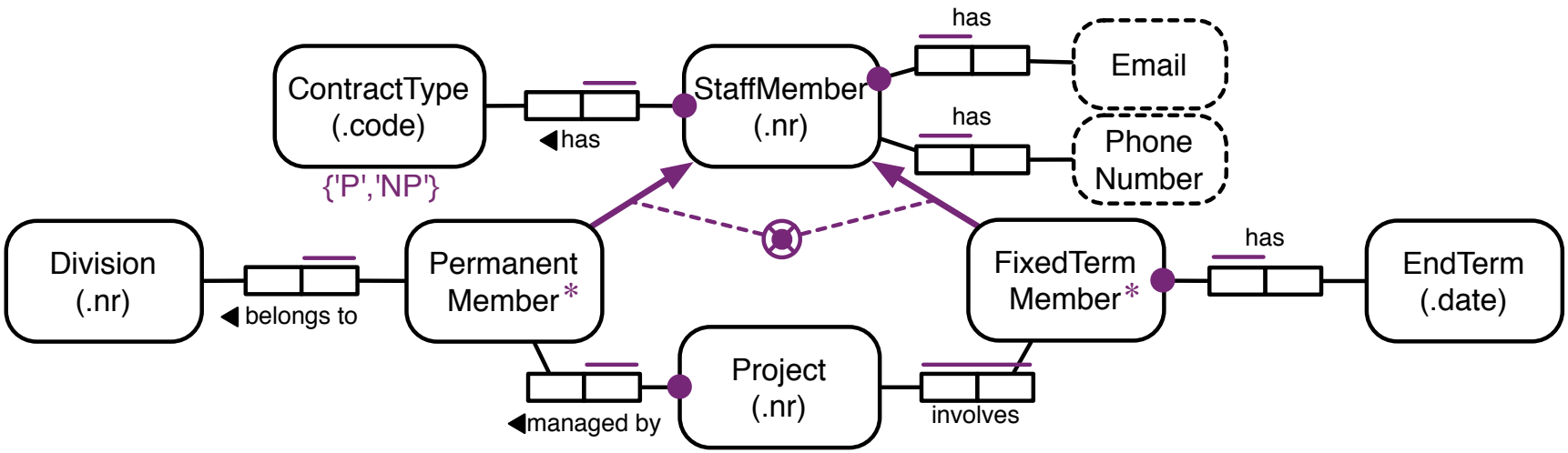
*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'

{'P','NP'}
StaffMember(staffNr, contrCode, eMail, [phoneNumber])

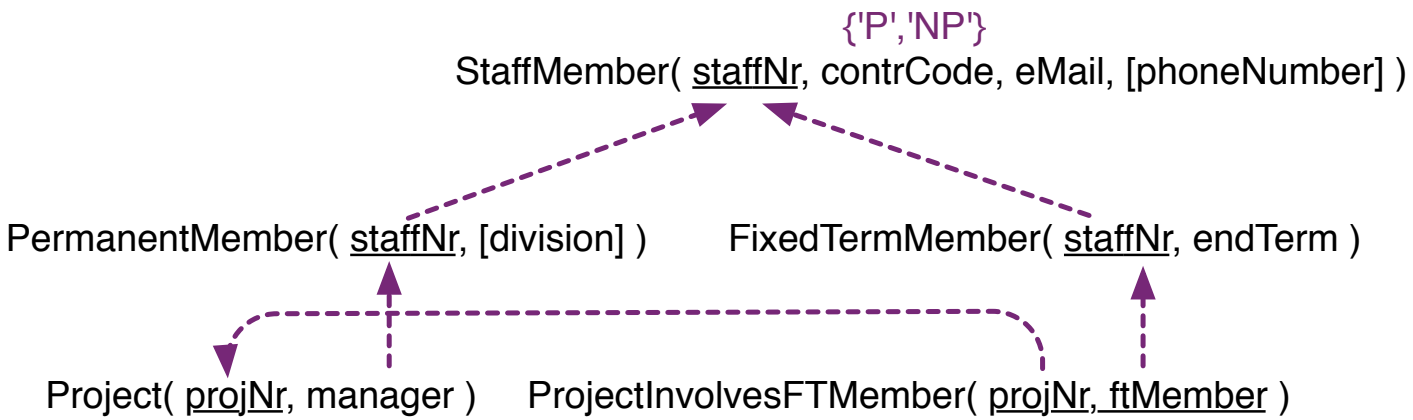
PermanentMember(staffNr, [division]) FixedTermMember(staffNr, endTerm)

Project(projNr, manager) ProjectInvolvesFTMember(projNr, ftMember)

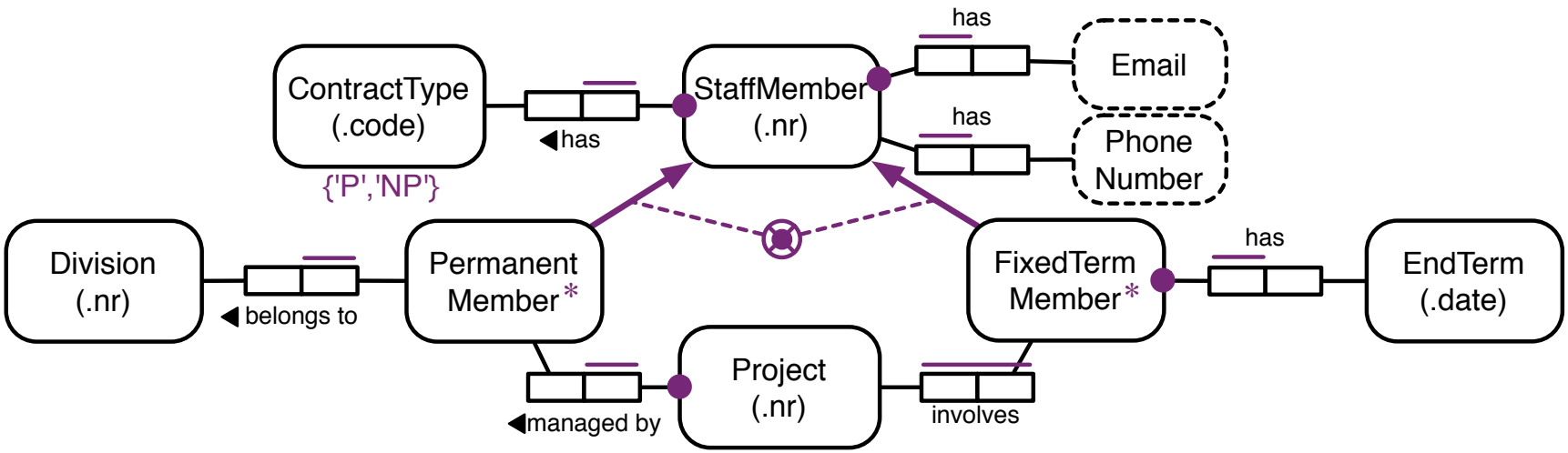
Separation - Representative Example



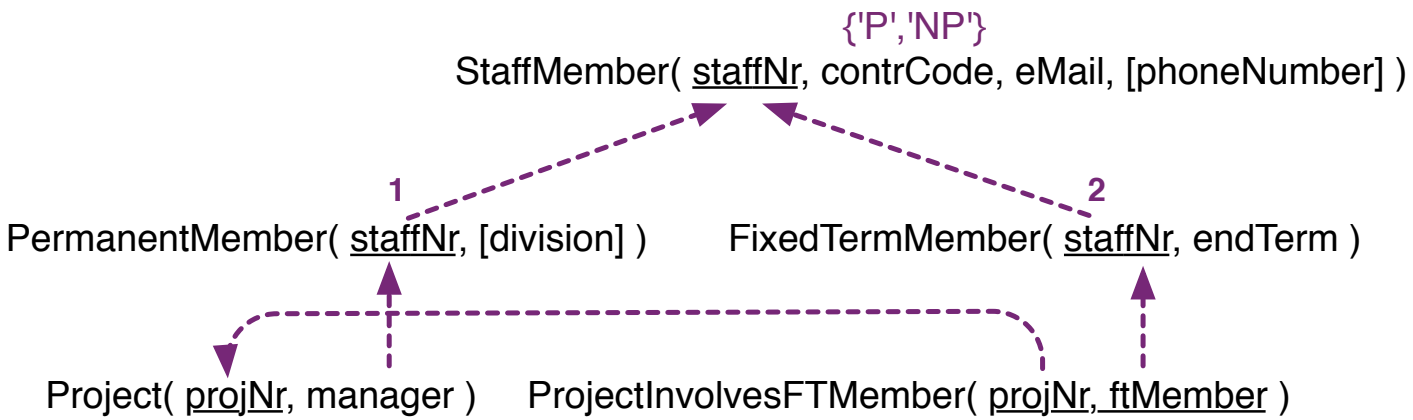
*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'



Separation - Representative Example



*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'



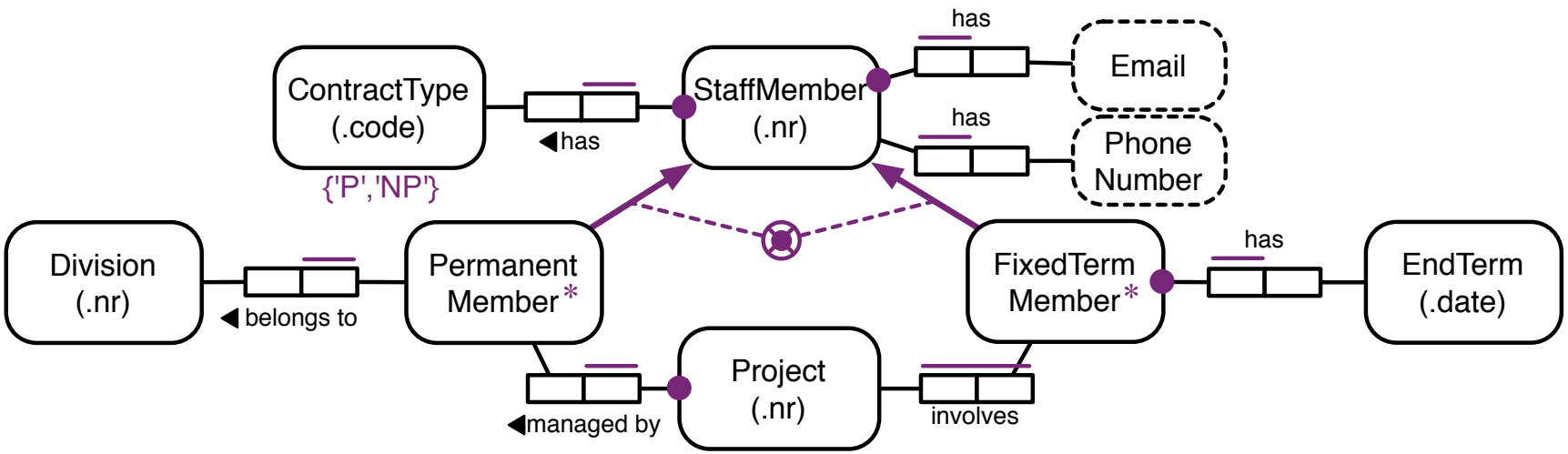
[1]**only where** contrCode = 'P' [2]**exactly where** contrCode = 'NP'

Partition

Applied when the subtypes are exclusive or form a partition.

- Idea: map only the subtypes to tables, reconstructing the supertype using unions.
- Roles attached to the supertype are virtually replicated and pushed down to each subtype, before applying the relational mapping.
- If the subtypes form a partition, only one table per subtype with functional roles is needed.
- If the subtypes are only exclusive, a "complementary" table is needed. If that A is supertype of the exclusive subtypes B and C, the partition is obtained by considering B, C and $A \setminus B \cup C$ (not recommended).
- An exclusion constraints between the PKs of the subtype tables ensures that each supertype individual is maintained only in one table.
- Subtype constraints ignored or trivially encoded, so as the discriminator entity type.
- Main advantage: null values minimization, fast queries for subtypes.
- Main weakness: unions needed for querying the superclass (a view can be constructed for this).

Partition - Representative Example



* **Each** PermanentMember **is a** Member **who** has ContractType 'P'
* **Each** FixedTermMember **is a** Member **who** has ContractType 'NP'

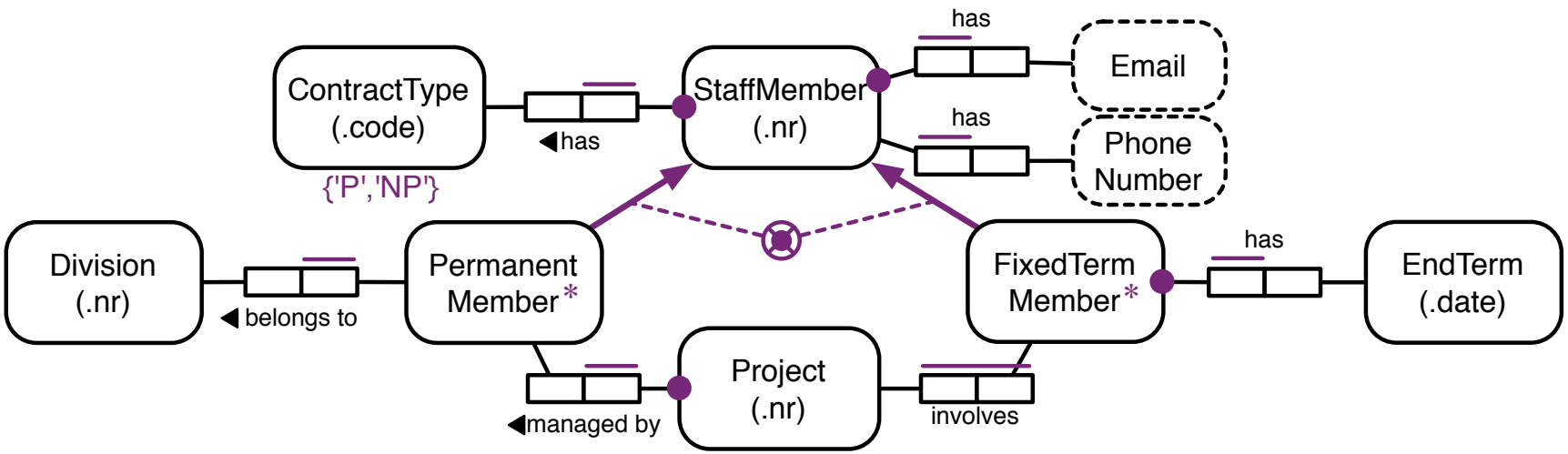
PermanentMember(staffNr, [division], eMail, [phoneNumber])

FixedTermMember(staffNr, endTerm, eMail, [phoneNumber])

Project(projNr, manager)

ProjectInvolvesFTMember(projNr, ftMember)

Partition - Representative Example



* **Each** PermanentMember **is a** Member **who** has ContractType 'P'
* **Each** FixedTermMember **is a** Member **who** has ContractType 'NP'

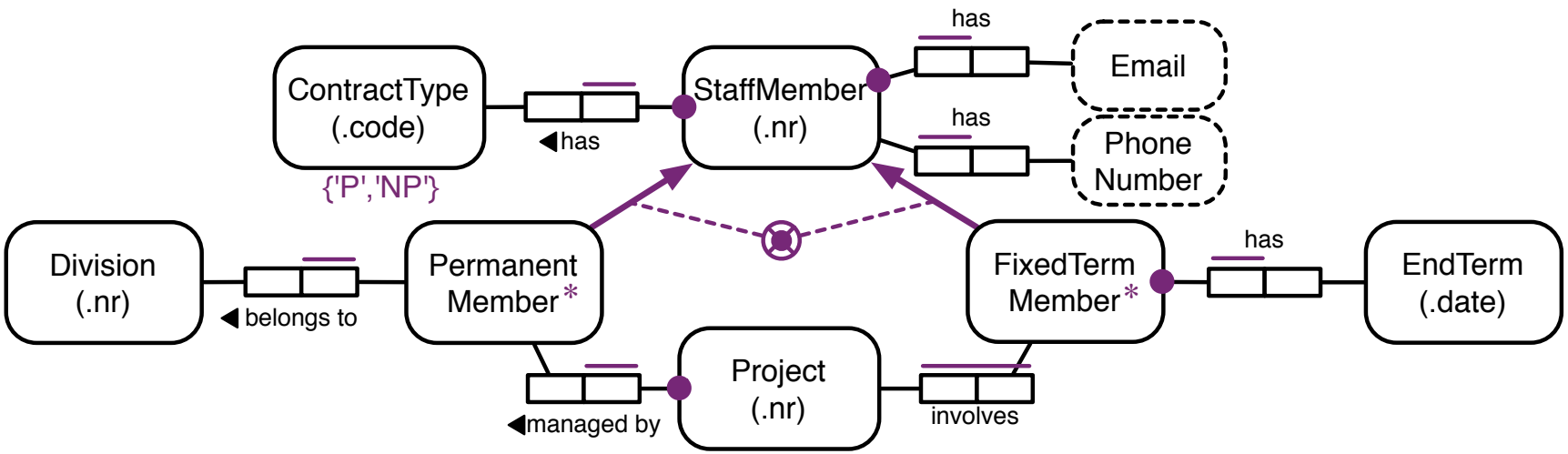
PermanentMember(staffNr, [division], eMail, [phoneNumber])

FixedTermMember(staffNr, endTerm, eMail, [phoneNumber])

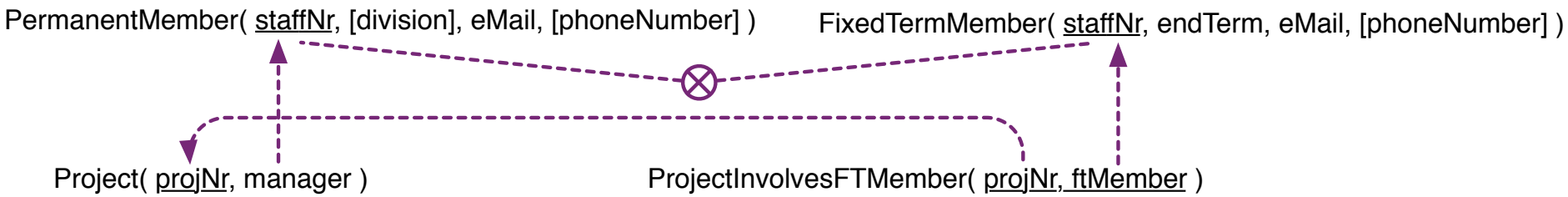
Project(projNr, manager)

ProjectInvolvesFTMember(projNr, ftMember)

Partition - Representative Example



*Each PermanentMember **is a** Member **who** has ContractType 'P'
*Each FixedTermMember **is a** Member **who** has ContractType 'NP'

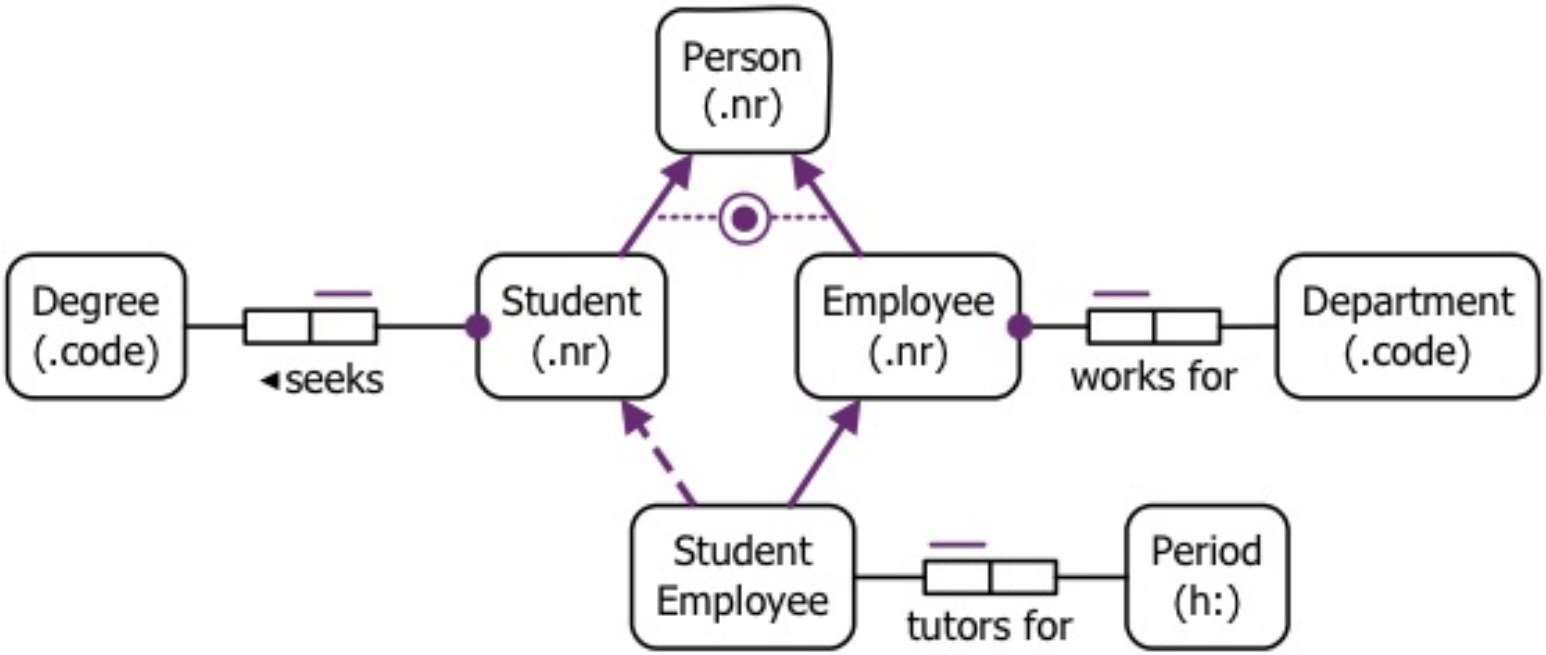



```
*StaffMember( staffNr ) =
    PermanentMember( staffNr ) union FixedTermMember( staffNr )
```

Mapping Hierarchies with Different Preferred Identifiers

- The root supertype table is obtained in the standard way.
- Each subtype overriding the preferred identification scheme is mapped separately.
- If the subtype has a corresponding total table, then the preferred reference of the supertype is added as a further column with a FK pointing to the supertype table.
- If the subtype has no total table, then an extra reference table is introduced to connect the identifiers of the subtype to identifiers of the supertype.
- Additional consistency constraints are introduced for multiple inheritance.

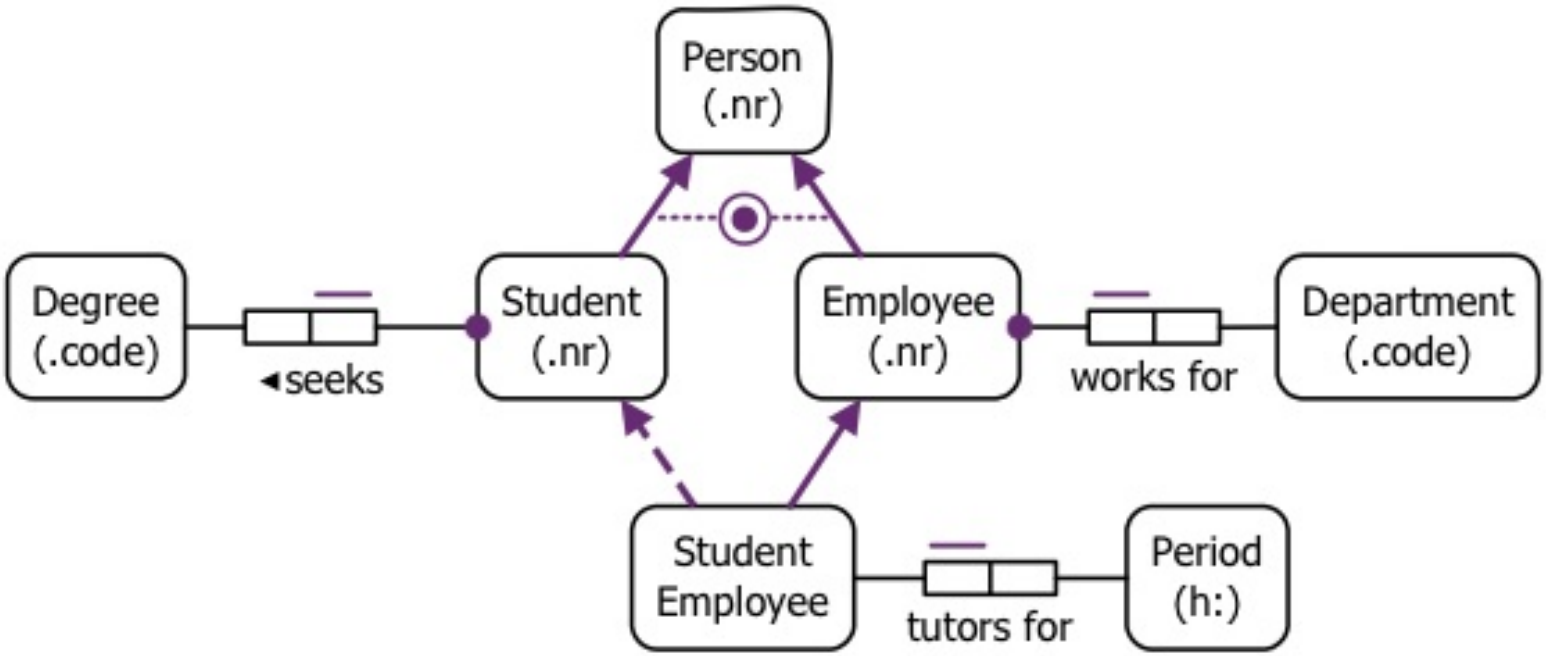
Hierarchies with Different Ids - Representative Example



- Student, Employee, Person mapped to separate tables.
- Student and Employee add a reference to Person in their total tables.
- StudentEmployee inherits Employee's preferred id $\rightarrow$ two choices:
  - Separation strategy. In this case the functional role "tutors for" is considered mandatory in the table, because it is the only distinctive feature of StudentEmployee.
  - Absorption strategy. Functional role "tutors for" added as optional column in Employee table. Discriminator column could be introduced.

Hierarchies with Different Ids - Representative Example

Separate mapping strategy for each object type of the hierarchy.



Student (studentNr, personNr, degree, ...)

Person (personNr, ...)

Employee (empNr, personNr, deptCode, ...)

StudentEmployee (empNr, tutePeriod, ...)

[1] **only where** personNr **in** Student.personNr

Rmap Procedure

0. Indicate any absorption-overrides (separation or partition) for subtypes, and absorb other subtypes into their top supertype.
   Mentally erase all explicit preferred identification schemes, treating compositely identified object types as "black boxes".
1. Map each fact type with a compound UC to a separate table.
2. Fact types with functional roles attached to the same object type are grouped into the same table, keyed on the object type's identifier.
3. Map 1:1 cases to a single table, generally favoring fewer nulls.
4. Map each independent object type with no functional roles to a separate table.
5. Unpack each "black box column" into its component attributes.
6. Map all other constraints and derivation rules.
   In case of absorption, subtype constraints on functional roles map to qualified optional columns, and those on nonfunctional roles map to qualified subset constraints.
   Nonfunctional roles of independent object types map to column sequences that reference the independent table.