Relational DB design

Denis Miginsky

Wares DB schema

```
CATEGORY:
  WARE TEXT
  CLASS TEXT

MANUFACTURER:
  RECIPE_ID INTEGER
  COMPANY TEXT

MATERIAL:
  RECIPE_ID INTEGER
  WARE TEXT
  AMOUNT INTEGER

PRODUCT:
  RECIPE_ID INTEGER
  WARE TEXT
  AMOUNT INTEGER
  PRICE REAL
```

Q: Why the schema should be exactly like this?

Q: Are there other schemas possible?

Q: How to choose the best one?

Another possible schema

DATA:

```
RECIPE_ID         INTEGER
COMPANY           TEXT
MATERIAL          TEXT
MATERIAL_CLASS    TEXT
MATERIAL_AMOUNT   INTEGER
PRODUCT           TEXT
PRODDUCT_CLASS    TEXT
PRODUCT_AMOUNT    INTEGER
PRODUCT_PRICE     REAL
```

Transformation query:

```
SELECT m.RECIPE_ID AS RECIPE_ID,
       m.COMPANY AS COMPANY,
       mt.WARE AS MATERIAL,
       mtc.CLASS AS MATERIAL_CLASS,
       p.WARE AS PRODUCT,
       …
FROM MANUFACTURER m
LEFT JOIN MATERIAL mt
ON m.RECIPE_ID=mt.RECIPE_ID
LEFT JOIN CATERGORY mtc
ON mt.WARE=mtc.WARE
INNER JOIN PRODUCT p
ON p.RECIPE_ID=m.RECIPE_ID
…
```

Problems with the alternative schema

- Table is much larger than ones in 4-table schema. Why shall we query all these rows to get just all the possible categories?
- How many new rows do we need to create a new recipe with the necessary stuff?
- 1-table schema allows inconsistencies.
- There are many duplicates in DB despite that all the rows are unique, technically. The DB requires much more space.

Database normalization

Normalization – the process of lossless transformation of DB schema (i.e. set of tables/relations) to reduce the data redundancy and improve the consistency and efficiency (usually).

Denormalization – the reverse process, i.e. injecting some data duplicates or derivative data to improve the performance.

Note: to reason about DB design it is necessary to take into the account all the possible data that could be in DB rather than current DB (since the DB is empty initially).

Superkey

Superkey – a subset of attributes such as for the given relation there are no tuples with the same values of them.

Alternatively – a simple projection that has the same cardinality as the original relation.

All the attributes of the relations are always form the superkey.

Primary and foreign keys

Candidate key – a superkey that has no proper subset of attributes that also form a superkey. I.e. minimal superkey.

There could be multiple candidate keys for the relation.

Prime attribute – an attribute that is a part of any candidate key

Primary key – one of the candidate keys chosen to be used as the main identifier.

Foreign key – for the given relation is a set of attributes which values are equal to attributes' values of some candidate key (usually, primary key) of any other relation, or possibly the same relation.
Representation of reference.

Natural and surrogate keys

Simple key – a candidate key that contains the only attribute

Compound key – a candidate key that contains multiple attributes

Natural key – a candidate key (simple or compound) that comes from the domain attributes

Surrogate key – a candidate key (usually simple one) that is artificially generated.
Usually it is used when there is either no natural key or natural key is too complex.

Surrogate key examples

```
--original schema, no natural key
MANUFACTURER:
  RECIPE_ID  INTEGER (PK)
  COMPANY  TEXT

--modified schema, complex natural key
CATEGORY:
  WARE      TEXT (CK)
  CLASS     TEXT (CK)
  ENTRY_ID INTEGER (PK)
```

Normalization criteria

Intuitive criterion: each domain constraint/rule is represented by the separate relation/table (or any other schema element in case of non-relational DB).

Examples:

- *Each recipe is owned by the particular company*
- *There are multiple materials (possibly 0)* ***and*** *multiple products for each recipe (2 rules)*

Formal criterion (for relational DB only): each relation/table in DB is in appropriate normal form.

1^{st} normal form (1NF)

A relation is in the **1^{st} normal form** (1NF) iff:

1. All values are atomic (i.e. single value of primitive type for each attribute and tuple combination)
2. A candidate key exists (i.e. all the tuples are unique)

This is the alternative definition of relation, i.e. **any** relation is in 1NF.

Functional dependency

?

remember the calculus…

2^{nd} normal form (2NF)

A relation is in the **2^{nd} normal form** (2NF) iff:

1. The relation is in 1NF
2. Any non-prime attribute is in **functional dependency** from the whole candidate key (for each key). There must be no functional dependencies from any proper subset of candidate key.

2NF violation

```
--modified schema, non-2NF

CATEGORY:
  WARE                 TEXT  (PK)
  CLASS                TEXT  (PK)
  RECOMMENDED_PRICE    REAL
```

```
--modified schema, 2NF

CATEGORY:
  WARE                 TEXT  (PK,FK)
  CLASS                TEXT  (PK)

WARE:
  WARE                 TEXT  (PK)
  RECOMMENDED_PRICE    REAL
```

Problem: RECOMMENDED_PRICE depends on WARE only, not CLASS

Properties of functional dependency

Let $\mathbf{R=\{A_1,\ldots,\ A_m,\ B_1,\ldots B_n\}}$ is **(m+n)**-ary relation.

If it can be represented as function **F:** $\mathbf{\{A_1,\ldots,\ A_m\}\rightarrow\{B_1,\ldots B_n\}}$,
then $\mathbf{\{A_1,\ldots,\ A_m\}}$ is a <u>superkey</u>.

If it can also be represented as **F':** $\mathbf{\{A_1,\ldots,\ A_{m-1}\}\rightarrow\{A_m,B_1,\ldots B_n\}}$,
then $\mathbf{\{A_1,\ldots,\ A_{m-1}\}}$ is a <u>superkey</u> and thus $\mathbf{\{A_1,\ldots,\ A_m\}}$ is <u>not a candidate key</u>.

Otherwise $\mathbf{\{A_1,\ldots,\ A_m\}}$ <u>is a candidate key</u>.

Heath’s theorem

Let **R={A, B, C}** is the relation with attributes **A**, **B**, **C** and **A→B**, then
R= {A, B} ⋈ {A, C}, where **{A, B}**, **{A, C}** are projections of **R.**

Lossless decomposition

The **lossless decomposition** – replacement of the original relation **R** with two or more projections, each one with proper subset of attributes of **R**, such as the join of all the projections gives exactly **R**.

By **join** here we mean equijoin on proper attributes and the projection/rename that removes join attributes duplicates.
It is very close to natural join, but allows to name attributes arbitrary and tune the join attributes as needed.

3^{rd} normal form (3NF)

A relation is in the **3^{rd} normal form** (3NF) iff:

1. The relation is in 2NF
2. For non-prime attributes there are no transitive functional dependencies from any candidate key

3NF violation

```
--modified schema, non-3NF

MANUFACTURER:
   RCEIPE_ID      INTEGER (PK)
   COMPANY        TEXT
   ADDRESS        TEXT
```

```
--modified schema, 3NF

MANUFACTURER:
   RECIPE_ID      INTEGER  (PK)
   COMPANY        TEXT     (FK)

COMPANY:
   COMPANY        TEXT     (PK)
   ADDRESS        TEXT
```

Problem: ADDRESS is in functional dependency from COMPANY, but not RECIPE_ID directly.

For all the rows with the same company the address will also be the same, that is redundant.

Boyce-Codd normal form (BCNF)

A relation **R** is in the **Boyce-Codd normal form** (BCNF, 3.5NF, Heath's normal form) iff for any functional dependency **A→B** between sets of attributes **A, B ⊆ R** at least one of the following conditions is true:

1. **A→B** is trivial, i.e. **B ⊆ A**
2. **A** is a **superkey**, i.e. a superset for some candidate key

Notes on BCNF

- Functional dependency from **superkey** means that the dependency is from some **candidate key** in fact (by definition of CK)

- All the candidate keys are in functional dependency from each other (also by the definition of CK).

- For BCNF there must be no functional dependencies for prime attributes (unless they form the whole candidate key/superkey – see previous note) from anything but the superkeys. This situation is not covered by 3NF

BCNF violation

```
--modified schema, non-BCNF

PRODUCT:
   RECIPE_ID    INTEGER (PK, CK)
   WARE         TEXT    (PK)
   BRAND        TEXT    (CK)
   …
```

```
--modified schema, BCNF

PRODUCT:
   RECIPE_ID    INTEGER (PK,FK)
   BRAND        TEXT    (PK,FK)
   …

BRANDING:
   BRAND        TEXT    (PK)
   WARE         TEXT
```

Domain constraint: each company has its own brand for each ware it producing. Possibly there are multiple brands for the same product with different recipes. Different companies must have different brands.

Problem: WARE (part of PK) depends on BRAND (non-prime). This is 3NF-complaint, because WARE is prime. But BCNF is violated.

More complex redundancy

```
--modified schema, BCNF

CATEGORY:
   WARE                  TEXT  (PK)
   CLASS                 TEXT  (PK)
   TRADER                TEXT  (PK)
   PROD_CONDITION        TEXT  (PK)
```

Domain constraint: There could be multiple traders for the same ware as well as multiple production conditions. So all the attributes are prime. And the relation is in BCNF.

Problem: there is definitely high redundancy. Intuitively both the trader and the condition depend on ware, but not on class. However, it is not a functional dependency.

Multivalued dependency

Let **A, B, C** ⊆ **R**. Multivalued dependency **A ↠ B** holds on **R** iff for each value of **(A, C)** the set of values of **B** in **R** depends only from **A**

SQL-like code:

```
SELECT B FROM R WHERE A=a AND C=c1
SELECT B FROM R WHERE A=a AND C=c2
```

Both queries give the same result with multiple rows in general case.

Lemma: Let **R={A, B, C}** is the relation with attributes **A**, **B**, **C**.
In this case **A ↠ B** means also that **A ↠ C**
This can be written as **A ↠ B | C**

Fagin’s theorem

Let **R={A, B, C}** is the relation with attributes **A**, **B**, **C**.
R= {A, B} ⋈ **{A, C}** iff **A ↠ B | C**.

The same is Heath’s theorem, but for multivalued dependency

4^{th} normal form (4NF)

A relation **R** is in the **4^{th} normal form** (4NF, Fegin's normal form) iff:

1. **R** is in BCNF
2. **A ↠ B** holds on **R** means one of the following:
   a) **A → B** (and A is a superkey according to BCNF)
   b) **A ↠ B** is trivial, i.e. either **B ⊆ A** (and thus **A → B**), or **A ∪ B = R** (and thus the only candidate key)

Normalized multivalued dependencies

```
--modified schema, non-4NF

CATEGORY:
   WARE                    TEXT  (PK)
   CLASS                   TEXT  (PK)
   TRADER                  TEXT  (PK)
   PROD_CONDITION          TEXT  (PK)
```

```
--modified schema, 4NF

CATEGORY:
   WARE                    TEXT  (PK)
   CLASS                   TEXT  (PK)

TRADER:
   WARE                    TEXT  (PK)
   TRADER                  TEXT  (PK)

PROD_CONDITION:
   WARE                    TEXT  (PK)
   CONDITION               TEXT  (PK)
```

Non-binary dependency

```
--4NF

RETAIL:
   WARE                       TEXT  (PK)
   TRADER                     TEXT  (PK)
   MANUFACTURER               TEXT  (PK)
```

Domain constraint: if

- the trader has a license to sell the ware,
- the manufacturer produces this ware,
- and the trader has a retail contract with the manufacturer,

then the trader **will** sell the given ware of the given manufacturer.

Join dependency

Let **A, B, … Z** ⊆ **R**. Join dependency ***(A, B, … Z)** holds on **R** iff

$$\mathbf{R} = \bowtie_{\beta=A\ldots Z} \beta$$

***(X, Y)** means **A ↠ B | C**, where **X = A ∪ B**, **Y = A ∪ C**, i.e. join dependency is the generalized version of multivalued dependency.

5^{th} normal form

A relation **R** is in the **5^{th} normal form** (5NF, project-join normal form, PJNF) iff (very informally) there are no join dependencies besides multivalued dependencies allowed by 4NF.

Normalized non-binary dependency

```
--non-5NF

RETAIL:
   WARE                  TEXT  (PK)
   TRADER                TEXT  (PK)
   MANUFACTURER          TEXT  (PK)
```

```
--5NF

RETAIL_LICENSE:
   TRADER                TEXT  (PK)
   WARE                  TEXT  (PK)

MANUFACTURING:
   WARE                  TEXT  (PK)
   MANUFACTURER          TEXT  (PK)

RETAIL_CONTRACT:
   TRADER                TEXT  (PK)
   MANUFACTURER          TEXT  (PK)
```

Domain constraint: if

- the <u>trader</u> has a license to sell the <u>ware</u>,
- the <u>manufacturer</u> produces this <u>ware</u>,
- and the <u>trader</u> has a retail contract with the <u>manufacturer</u>,

then the <u>trader</u> **will** sell the given <u>ware</u> of the given <u>manufacturer</u>.

DB design: practice

Problem statement elements:

- Problem domain
- Domain data (typically not in the relational form)
- Domain constraints
- Questions that come from the problem definition

Solution:

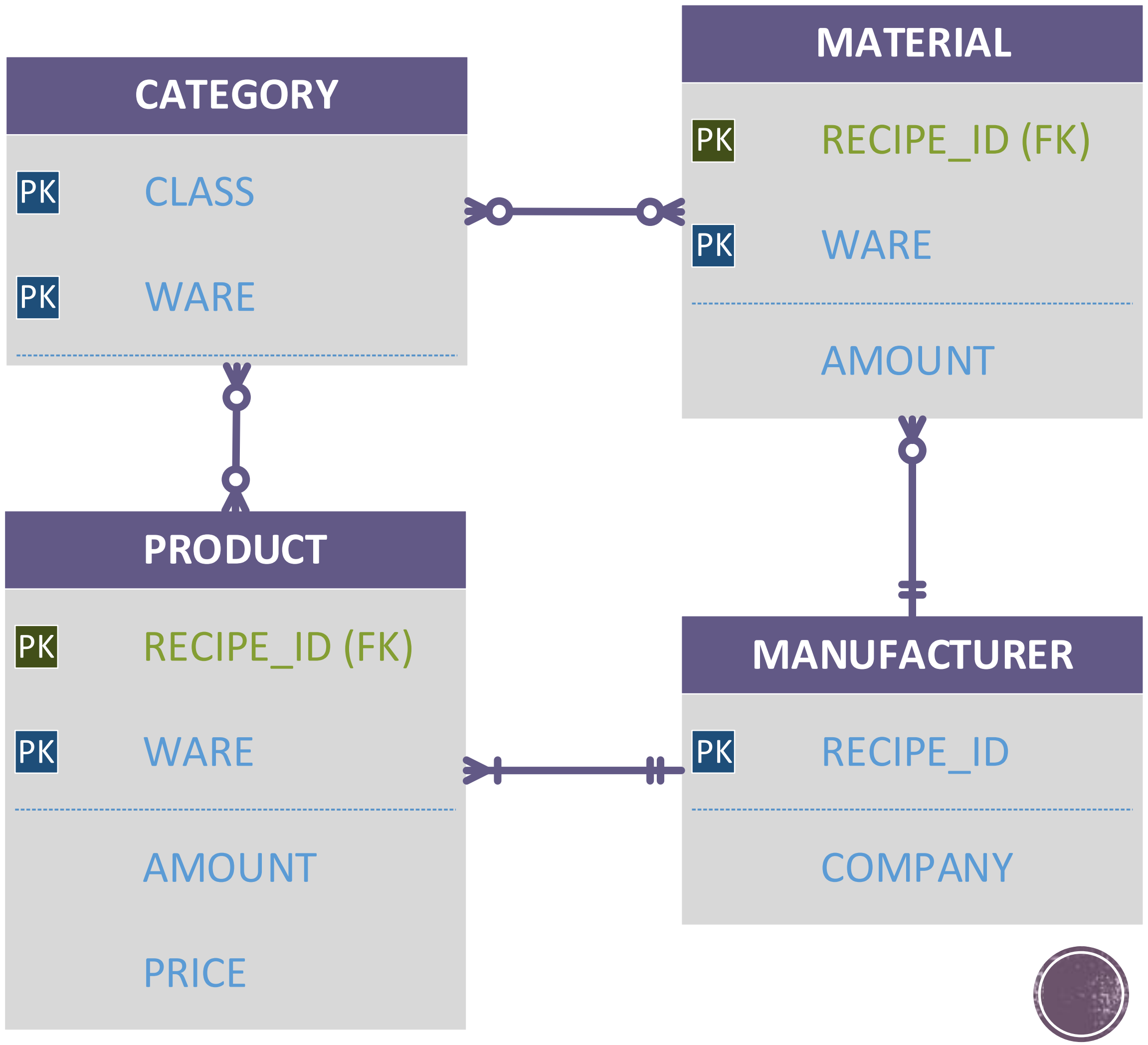
1. Normalized relational schema (or not relational) representing all the necessary data and as much domain constraints as it can.
2. Queries for the problem and optimizing modifications for the schema.

Entity-relationship (ER) diagrams: Crow's foot notation

Many-to-one (0…∞ - 1)

Many-to-many (0…∞ - 0…∞)

At least one to zero or one (1… ∞ - 0…1)

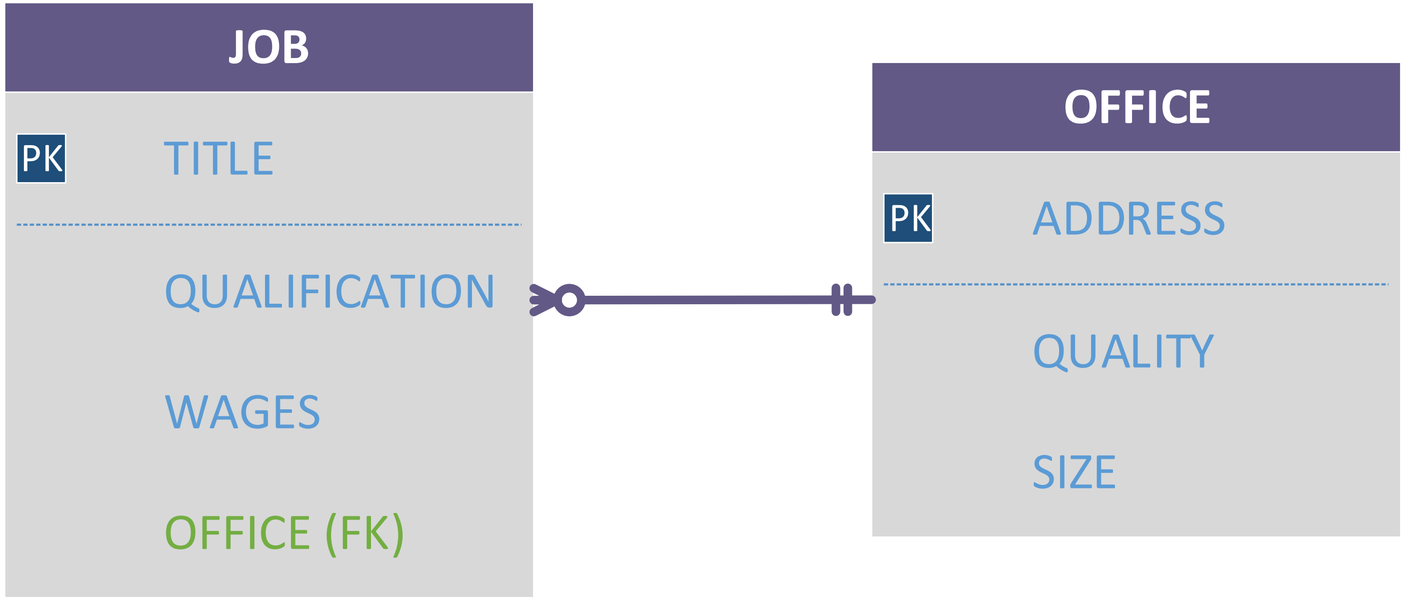


Schema design procedure

1. Identify **entities** and their primary keys (or construct surrogate keys). Each entity should be represented as separate table at first glance.
2. Identify all the **properties** of all the **entities**.
3. Each **single-valued property** (→) comes to the table of corresponded entity. Multiple → from **PK** are allowed inside the same table.
4. Each **multivalued property** comes to the separate table (due to 4^{th} normal form).
5. All the **properties** that have their own properties become **entities** and shall be moved to separate tables (due to 2^{nd} normal form)
6. Identify all the **relationships** between entities…

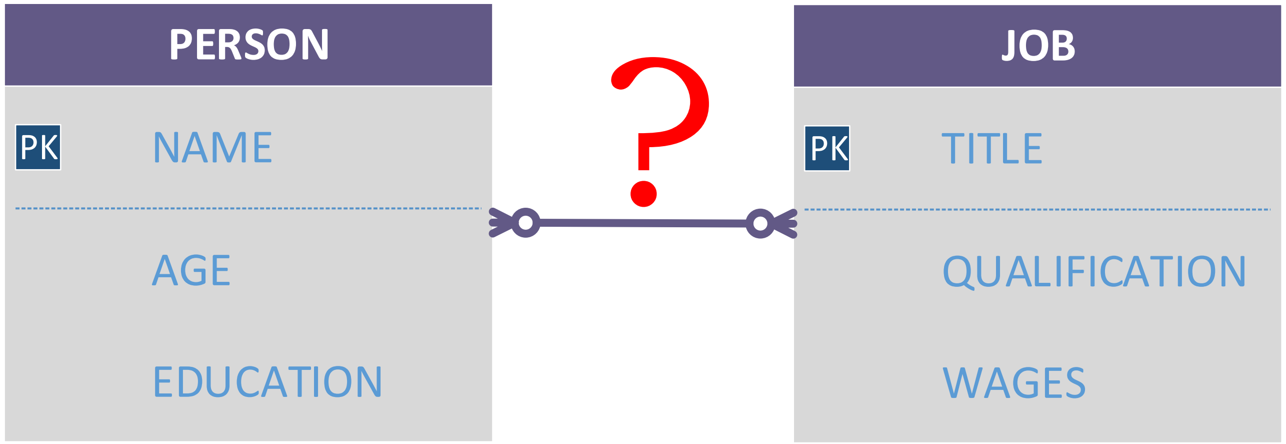
Relationships for entities: many-to-one

Many-to-one/functional dependency: the same as the single-valued property, represented as foreign key to the appropriate entity table (according to BCNF).



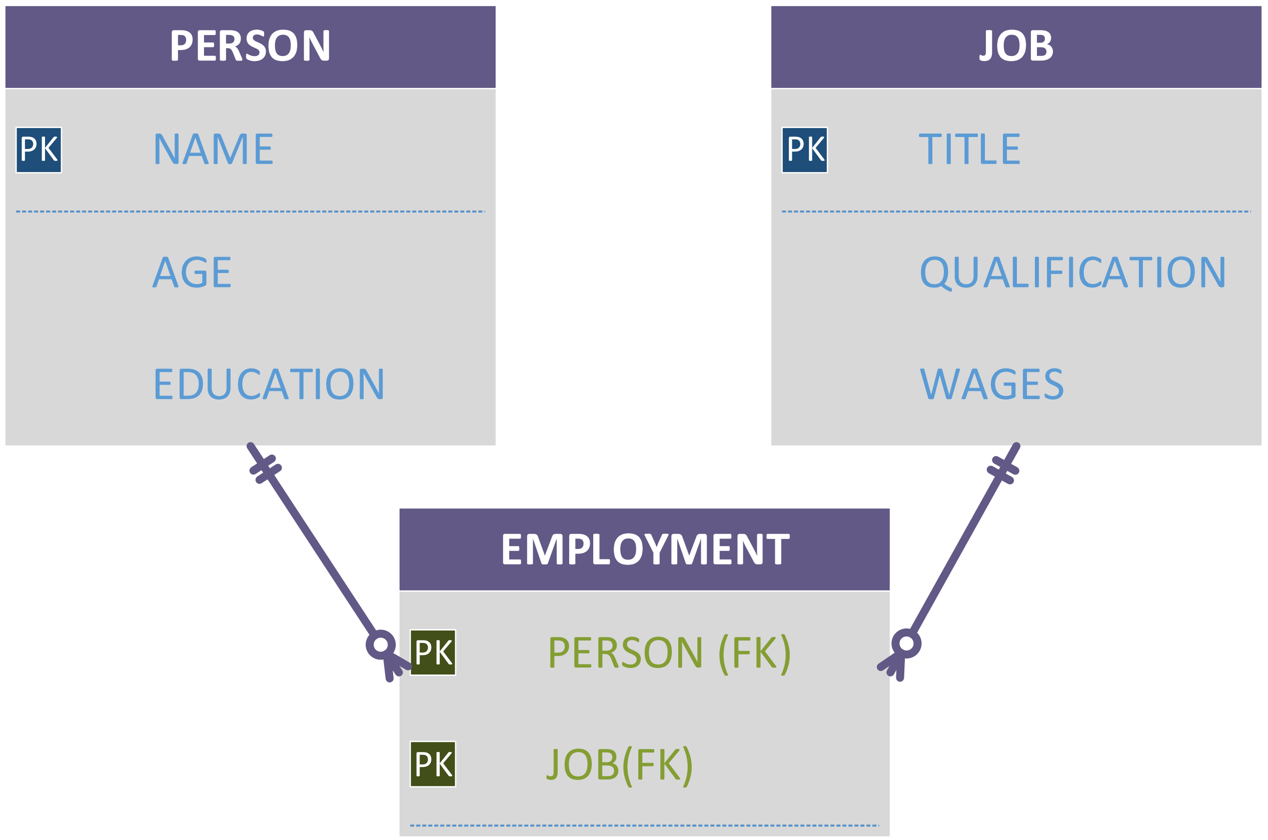
Relationships for entities: many-to-many

Many-to-many/multivalued dependency: the additional table is required to represent this constraint (according to 4NF).

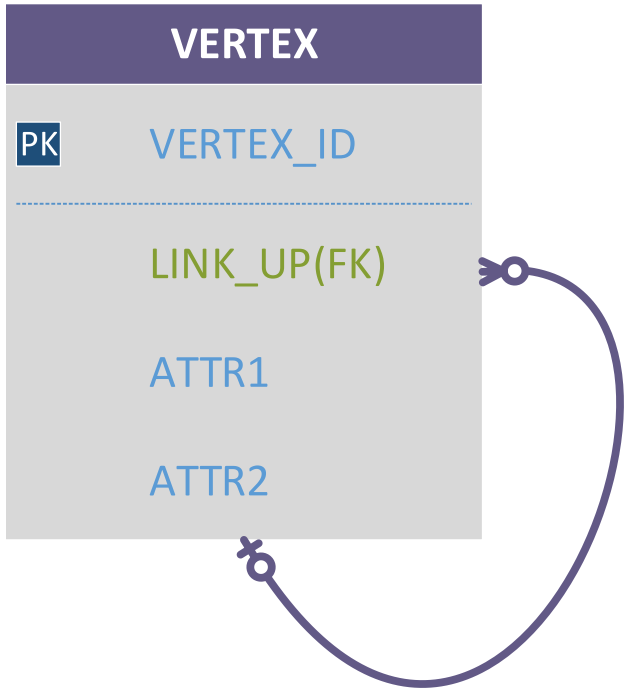


Many-to-many: solution

Many-to-many for entities is like **many-to-one** + **one-to-many**



Example: trees



Example: property graphs