Relational translation-II (Relationships)

Knowledge objectives

1. Enumerate the different possibilities to enforce integrity constraints in a database
2. Explain the problem generated on dropping a table from the logical schema, even if this does not contain other attributes than the PK
3. Enumerate two possibilities to replace a table in the logical schema, when this does not have other attributes than the PK
4. Enumerate the three criteria to choose the PK on fusing two tables linked by a 1-1 association
5. Explain the two possible implementations of symmetric reflexive associations in a RDBMS
6. Remember where to place the attributes of the UML associations when they are implemented on a RDBMS, depending on their multiplicity
7. Distinguish an FK with two attributes from two FKs of one attribute
8. Distinguish association classes that appear just due to UML syntax constraints from those truly association classes
9. Enumerate ten pros and cons of storing multivalued attributes either per row or per column

Understanding objectives

1. Solve a deadlock in the definition of FKs
2. Solve a deadlock in the loading of tables with FKs
3. Translate from a UML class diagram (with around 10 classes, some maybe associative classes, and related by associations, generalizations and aggregations) into an SQL schema

Application objectives

1. Choose and justify the best option to translate from a UML class diagram (with less than 10 classes, some maybe associative classes, related by associations, generalizations and aggregations) into an SQL schema, given the statistics of participation of the instances in the relationships and the queries
2. Given a multivalued attribute and an explanation of its usage, choose and justify the best option to implement it in a RDBMS

Constraints

Candidate keys

Foreign keys

Implementing constraints

1. In the CREATE TABLE sentence

   Usually available, efficient, automatic and internal

2. Assertions

   Efficient, automatic and internal

LOGICAL DESIGN

PHYSICAL DESIGN

3. Persistent Stored Modules
   1. Triggers

      Automatic and internal

   2. Procedures/functions

      Internal

4. Call Level Interface (e.g., ODBC, JDBC)

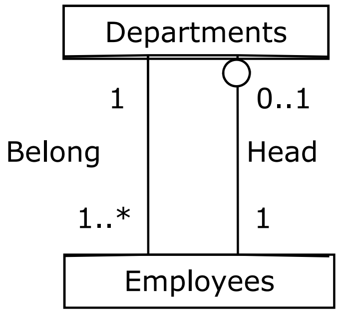
   Always available

Candidate keys

- Primary
  - May not be available in our DBMS (really rare)
  - Physically, generate a B-tree index

- Alternative
  - Are not part of standard SQL
  - Can be implemented by NOT NULL + UNIQUE
    - Physically, generate a B-tree index

Deadlock in the definition of FK



Dept(dpt,...,head)

Empl(dni,...,dpt)

A. Modify the tables
   1) Create "Dept" without FK
   2) Create "Empl" with FK
   3) Alter "Dept" adding the FK

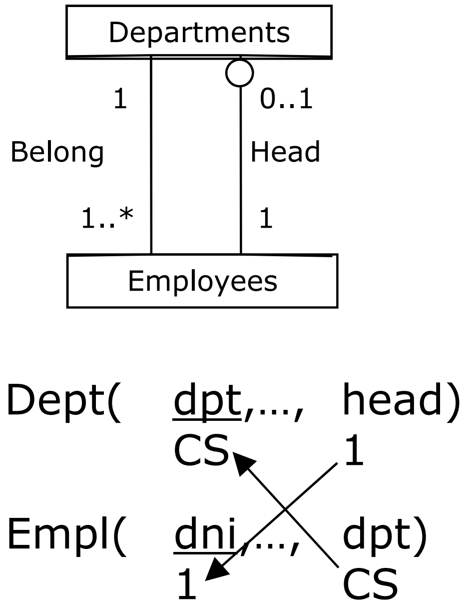
B. Create three tables
   1) Create "Dept" (it doesn't have FK)
   2) Create "Empl" with its FK
   3) Create "Head" with two FK

Dept(dpt,...)

Head(dpt,empl)

Empl(id,...,dpt)

Deadlock in the load of FK



A. Drop/Disable FK
   1) Alter "Dept" to drop/disable FK
   2) Insert the department
   3) Insert all employees
   4) Alter "Dept" to add/enable FK

   SET CONSTRAINT <name> [ENABLE|DISABLE];

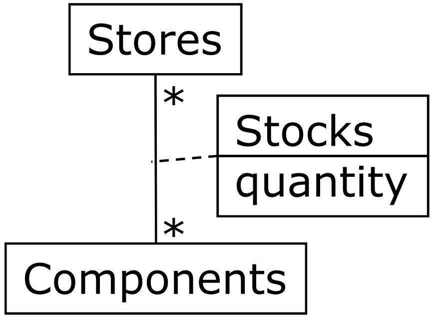
B. Defer the FK checking

   SET CONSTRAINTS <name> [IMMEDIATE|DEFERRED];

https://www.postgresql.org/docs/current/sql-set-constraints.html

Relavance of FK

Let's suppose that none of both classes has attributes



Stores(<u>city</u>)

Stocks(<u>store</u>,<u>comp</u>,quantity)

Components(<u>code</u>)

Even if tables only contain the identifiers,
they are still useful to enforce integrity constraints

A table with few instances can be replaced by an enumerated data type or check constraint

Binary associations

Multiplicities

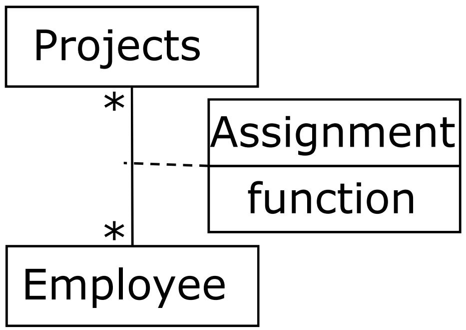
Maximum multiplicity:

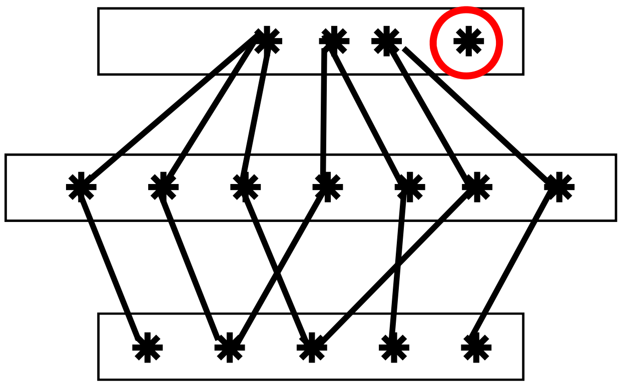
- Question:
  - Each instance in one end, how many can have <u>at most</u> in the other end?
- Possibilities for binary associations:
  - *-*
  - 1-*
  - 1-1

Minimum multiplicity:

- The three cases above split into subcases
- Question:
  - Could zeros exist (<u>possible</u> no participation of an instance in the association)?
    - If there are zeros, do they give rise to nulls in the relational translation?

Binary association (*-*)





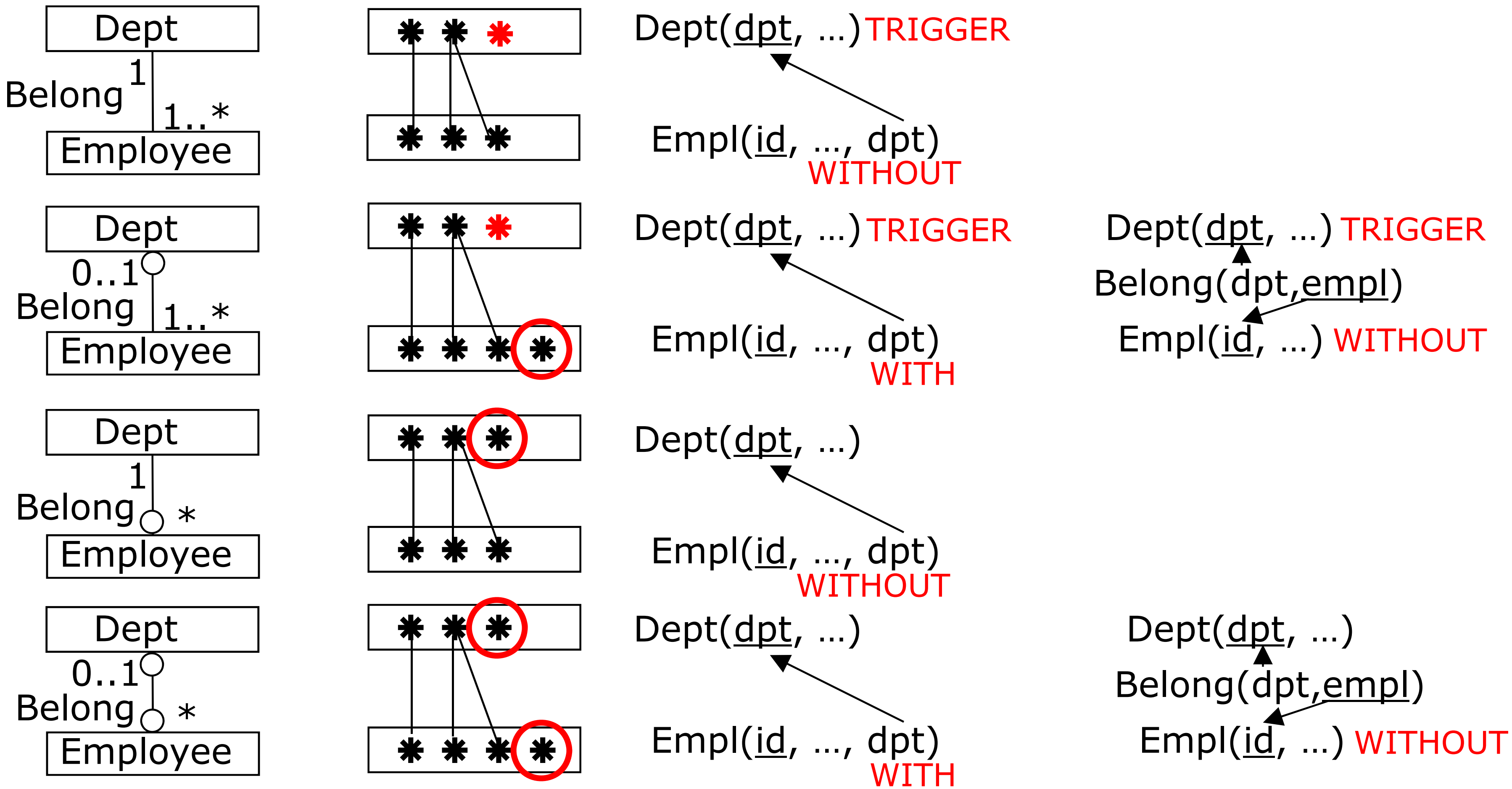
Projects(<u>pro</u>, …)

Assig(<u>empl</u>,<u>pro</u>,func)

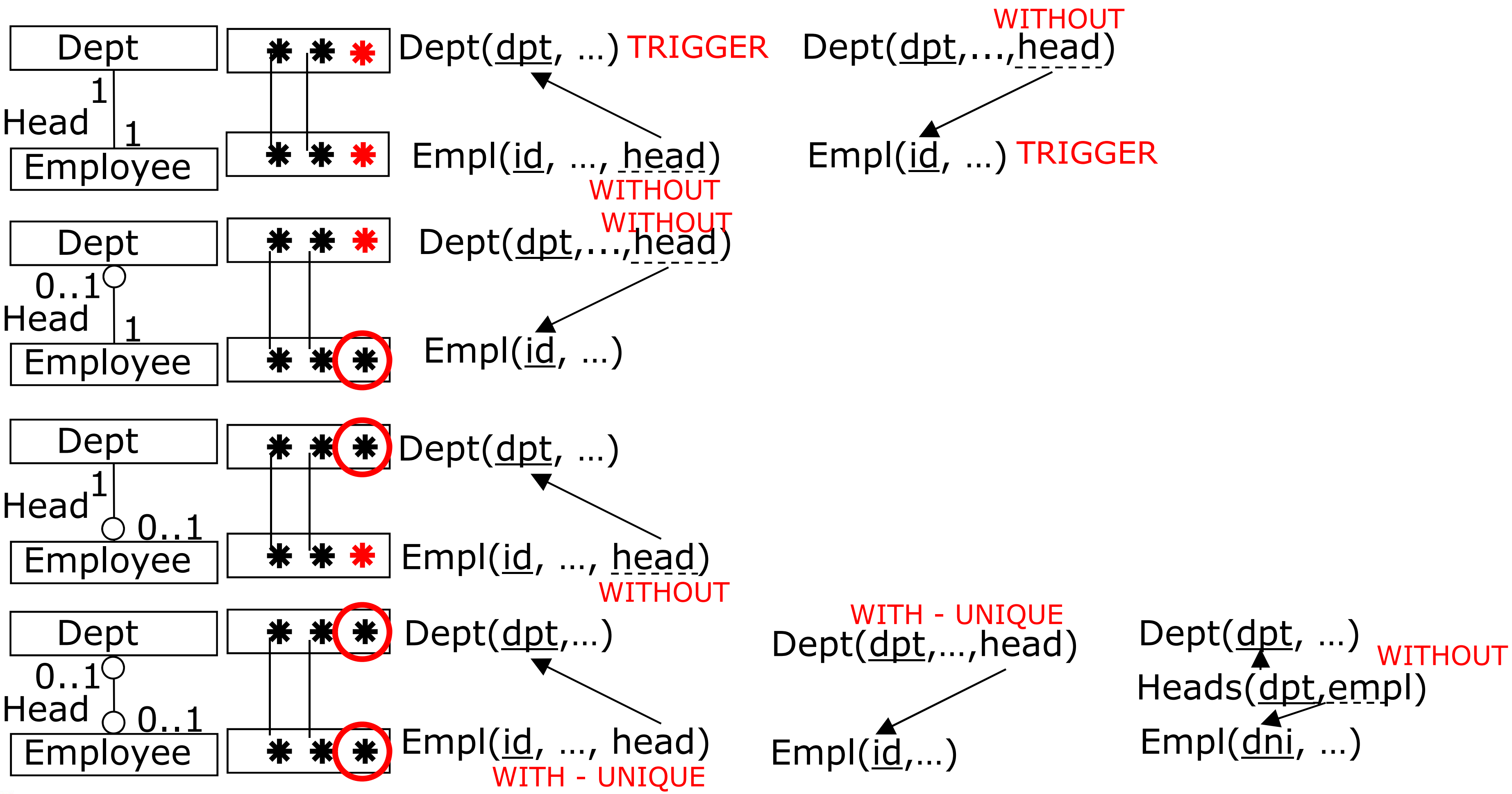
Employee(<u>id</u>, …)

Always a new table!!!

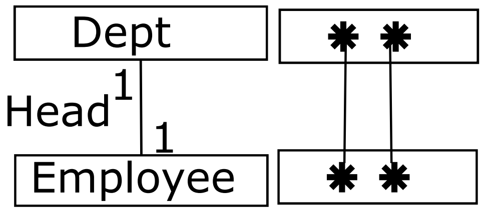
Binary associations (1-*)



Binary associations (1-1)



Fusing classes/Choosing PK



(also possible in 0..1 cases)

Dept(<u>dpt</u>,...)

Empl(<u>id</u>, ..., dpt) WITHOUT

Dept_Empl(<u>dpt</u>,...,empl,...)

We have to choose the primary key among the candidate keys

The choice depens on:

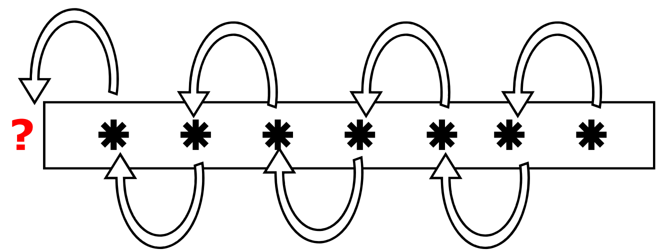
- Frequency of use in the queries
- Space required
- Frequency of change

Country_President(	country, ...,	president, ...)
	USA	B. Obama
	Spain	M. Rajoy

Reflexive associations

Reflexive associations

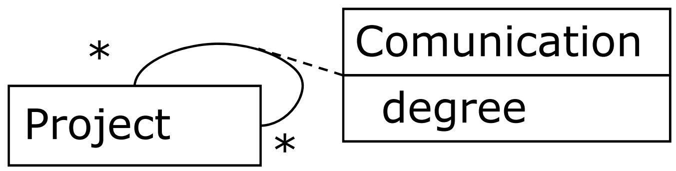
- Valid multiplicities:
  - *-* (Relatives)
  - 1-* (Mother)
  - 1-1 (Couple)
- Singularities:
  - Almost always have zeros
  - May be symmetric or not



Brother1	Brother2
John	Peter
Peter	John

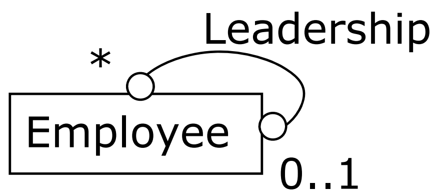
Friend1	Friend2	Grade
John	Peter	10
Peter	John	2

Reflexive multiplicities



Project(pro, …)

Requires_com(pro, pro-com, degree)

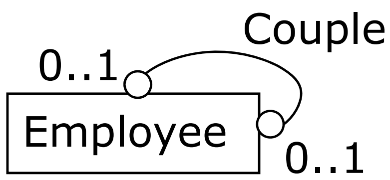


Employee(emp, …, leader) WITH

or

Employee(emp, …)

Leadership(emp, emp-leader) WITHOUT



Employee(emp, …, couple) WITH

or

Employee(emp, …)

Couple(emp, emp-couple) WITHOUT

Symmetric reflexive associations

- We must enforce the property
- Triggers may bring a satisfactory solution:

a) Storing both pairs

ON INSERT (a, b) → INSERT (b, a)
ON DELETE (a, b) → DELETE (b, a)
ON UPDATE…

a) Storing only one of the pairs

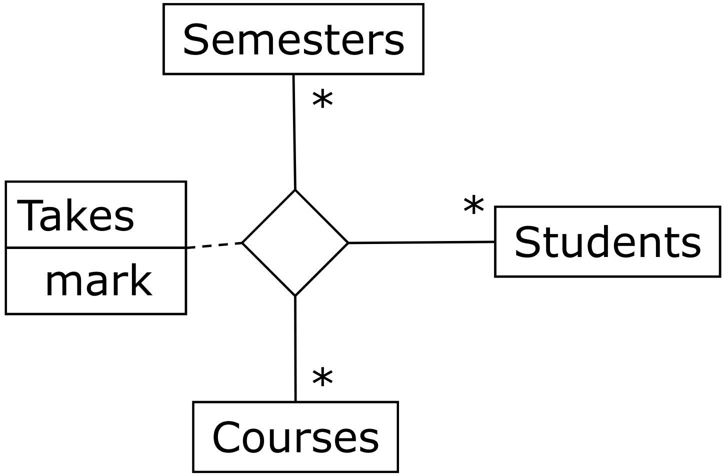
Simulate the whole set of pairs with a view:

```
CREATE VIEW v_couples AS
SELECT emp1, emp2 FROM couples
UNION
SELECT emp2, emp1 FROM couples;
```

ON INSERT (a, b) → if (b, a) already present, raise exception
ON DELETE (a, b) → if (b, a) is present instead, delete that
ON UPDATE…

N-ary associations

Ternary associations (*-*-*)



Students(<u>st</u>, …)

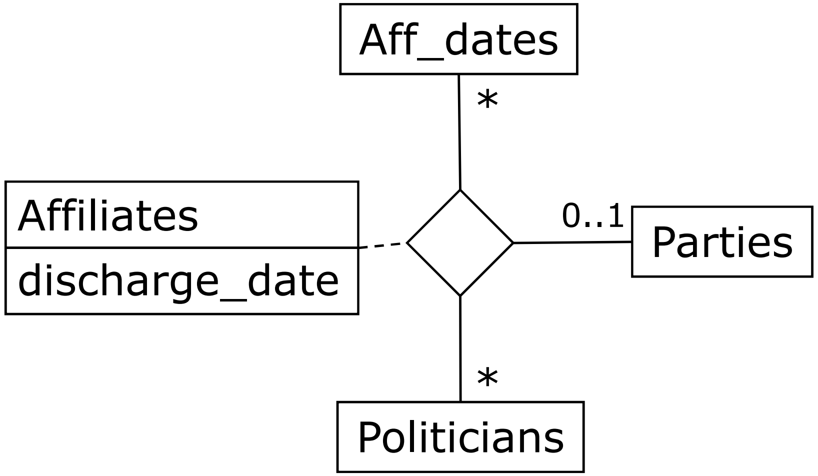
Semesters(<u>sem</u>, …)

Courses(<u>cou</u>, …)

Takes(<u>st, sem, cou</u>, mark)

Always a new table!!!

Ternary associations (*-*-1)



Parties(acronym, ...)

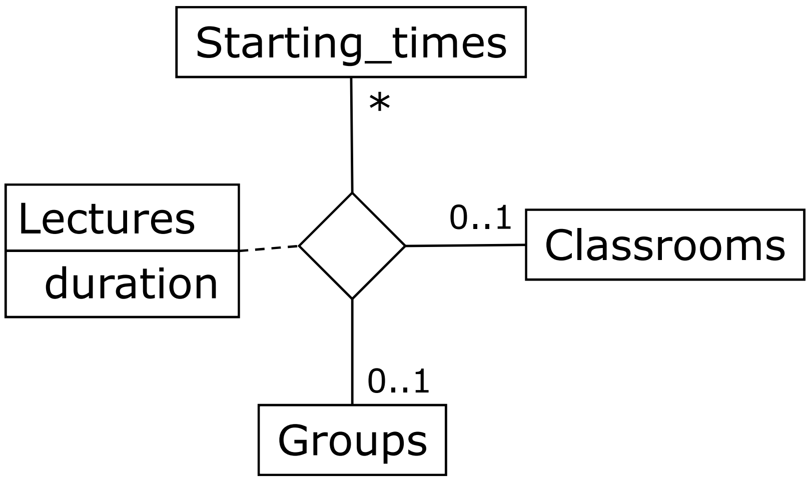
Aff_dates(date, ...)

Politicians(name, ...)

Affiliates(par, date, pol, dis)

Always a new table!!!

Ternary associations (*-1-1)



Groups(id, …)

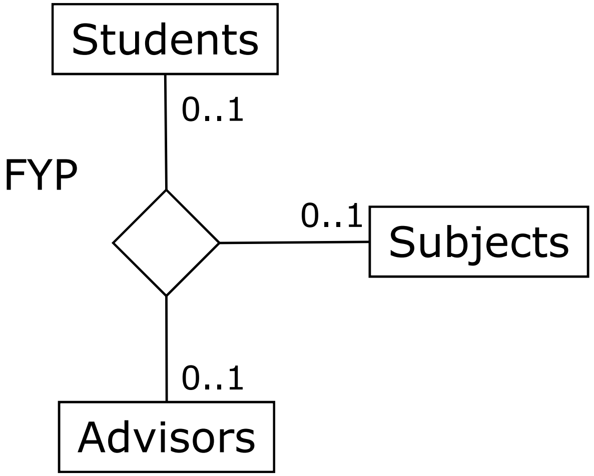
Starting_times(time, …)

Classrooms(id, …)

Lectures(group, time, clr, duration)

Always a new table!!!

Ternary associations (1-1-1)



Subjects(<u>subjects</u>, …)

Students(<u>name</u>, …)

Advisors(<u>name</u>, …)

FYP(subj, student, advisor)

Always a new table!!!

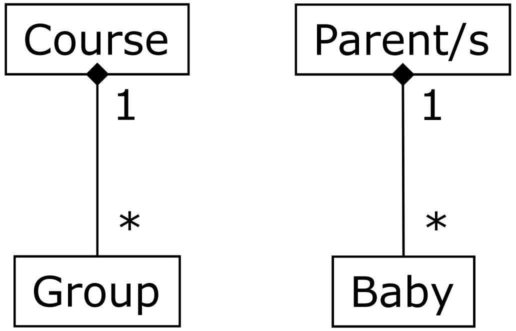
N-ary associations

- Binary: A new table or foreign key
- Ternary: A new table
- Quaternary: A new table
- …

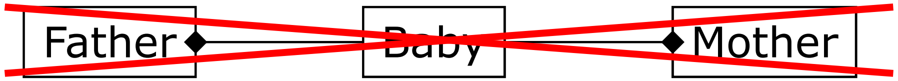
Compound aggregation

Compound aggregation (I)

- Weak class, with regard to the external key of the classical relational model

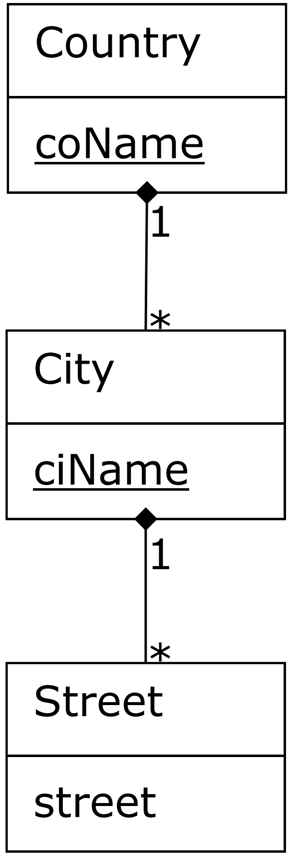


- A given class cannot be part of two



- Composition cannot have zeros at “to-one” side

Compound aggregation (II)



Country(coName, …)

City(coName, ciName, …)

MULTIATTRIBUTE FK

Street(country, city, street, …)

Multiattribute foreign keys

Accepted

City(country, city, ...)
France Paris
Spain Madrid

Street(country, city, street, ...)
France Madrid

FK violation

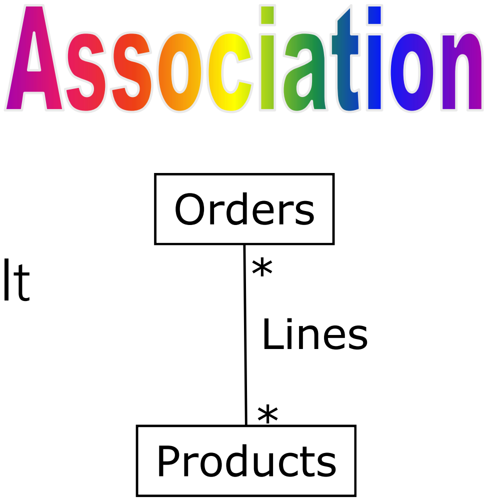
City(country, city, ...)
France Paris
Spain Madrid

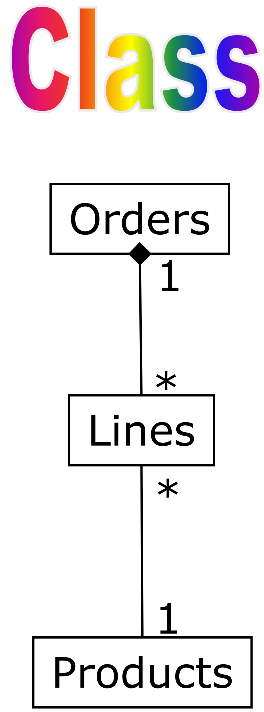
Street(country, city, street, ...)
France Madrid

Association classes

Association vs class

- Association (common)
  - Identified through its “legs”
  - Makes defining/checking constraints difficult
- Class (uncommon)
  - Identified by itself (it has an external key)
  - Makes defining/checking constraints easy





Attributes of associations

- *-* or n-ary (common)
  - In the table representing the association

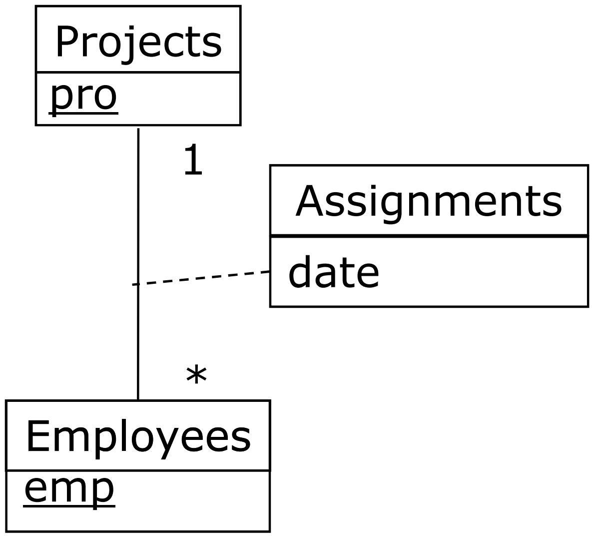
- 1-* (uncommon)
  - If any, in the table representing the association
  - Otherwise, in the table at the side of the star

- 1-1 (rare)
  - If any, in the table representing the association
  - If only one table (fusion), in it
  - Otherwise, in the table at the side of the zero (if any)

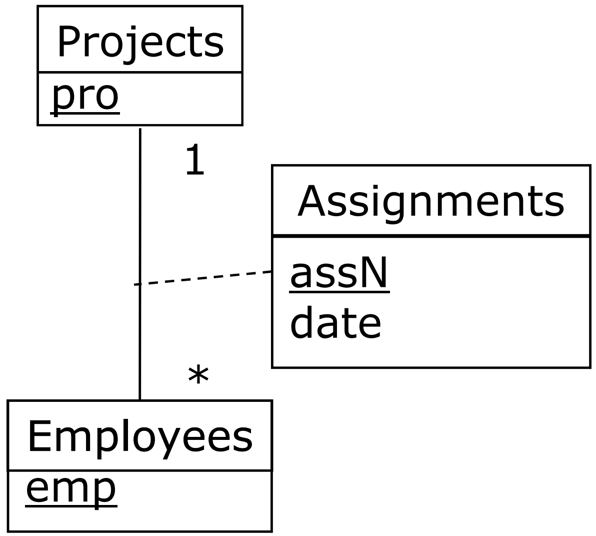
Always together with the foreign key

Association vs association class

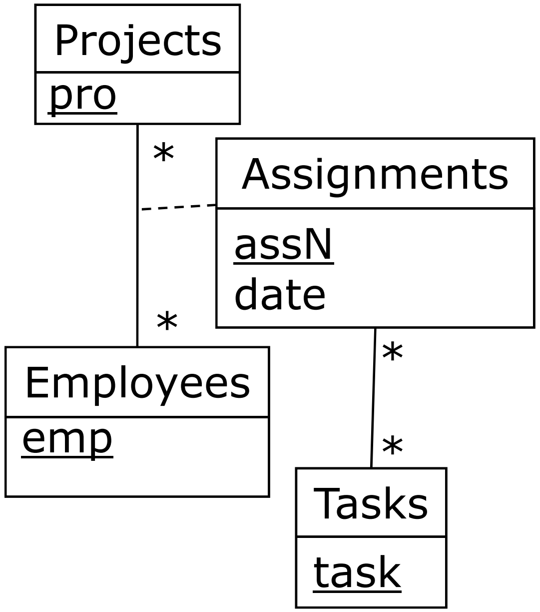
Association



Association class



Association classes (I)



Implemented as a table

Projects(pro, …)

Employees(emp, …)

Assignments(assN, emp, proj, date)

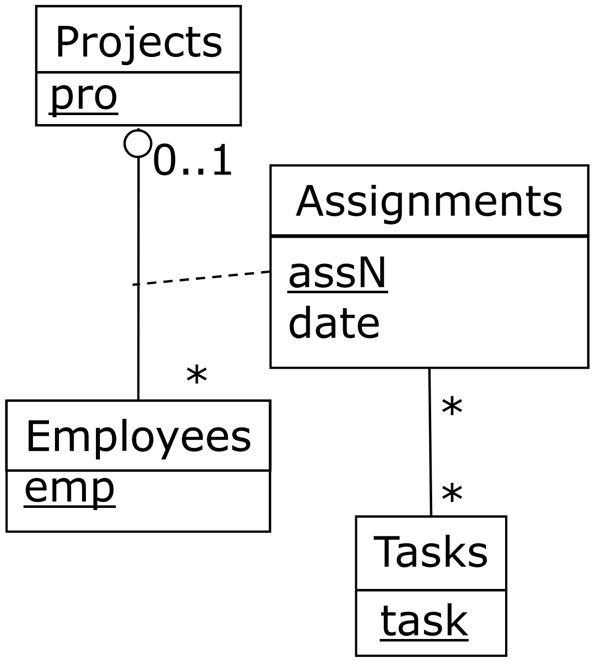
Task-Assig(task, assN)

Task(task, …)

FOREIGN KEY (assN) REFERENCES Assignments (assN)

Association classes (II)

The FK cannot point to "emp", because a task must be done by an employee who is assigned



Implemented as an FK

Projects(<u>proj</u>, …)

Employees(<u>emp</u>,… , proj, assN, date)

Task-Assig(<u>task, assN</u>)

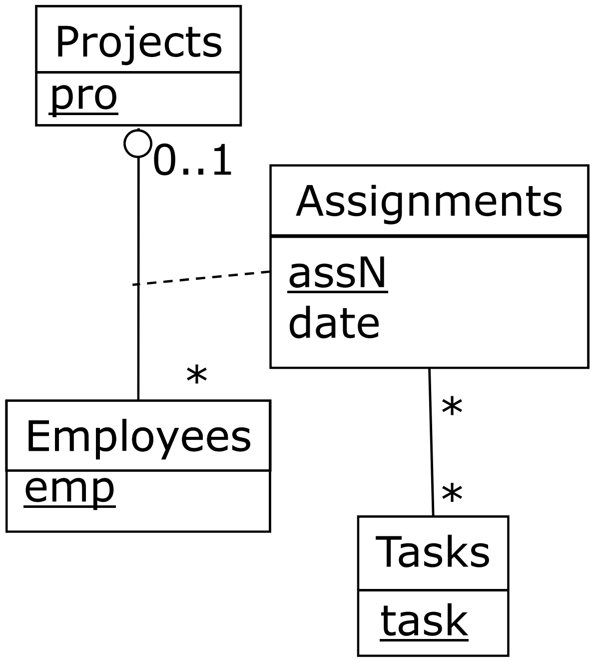
Tasks(<u>task</u>, …)

UNIQUE(assN)

CHECK ((proj IS NULL AND assN IS NULL) OR (proj IS NOT NULL AND assN IS NOT NULL))

Association classes (III)

The FK cannot point to "Employees", because a task must be done by an employee who is assigned



Implemented as a table

Projects(<u>proj</u>, …)

Employees(<u>emp</u>,…)

Assignment(proj, <u>emp</u>, assN, date)

Task-Assig(<u>task, emp</u>)

Tasks(<u>task</u>, …)

proj NOT NULL

Multivalued attributes

Per column

Per row

Multivalued attributes example (I)

Client
code: integer phone: string [*] ...

One value per column

Client(code,	office-phone,	secretary-phone,	cell-phone,	...)
C1	933333333	933333331	666666666	null

One value per row

Client(code, ...)

ClientTelephones(code,	place,	telephone)
C1	Office	933333333
C1	Secretary	933333331
C1	Cellular	666666666

Multivalued attributes example (II)

Experiment
id: integer measure: integer [*] ...

One value per column

Experiment(id, $\text{measure}_{\text{equipment1}}$, $\text{measure}_{\text{equipment2}}$, ..., $\text{measure}_{\text{equipmentn}}$)

id	$\text{measure}_{\text{equipment1}}$	$\text{measure}_{\text{equipment2}}$	...	$\text{measure}_{\text{equipmentn}}$
a	30	31		28

One value per row

Experiment(id, equipment, measure)

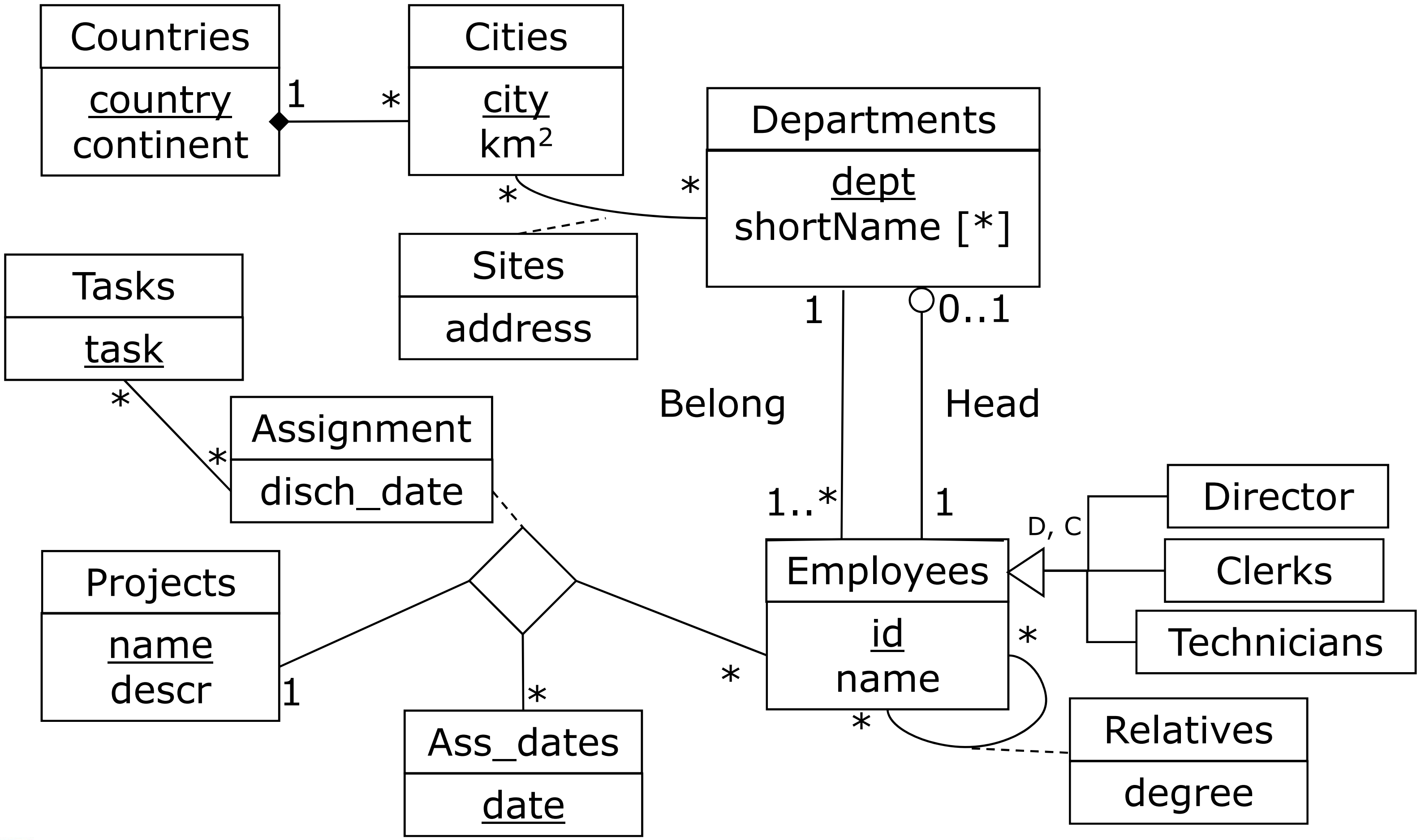
id	equipment	measure
a	1	30
a	2	31
...		
a	n	28

Multivalued attributes comparison

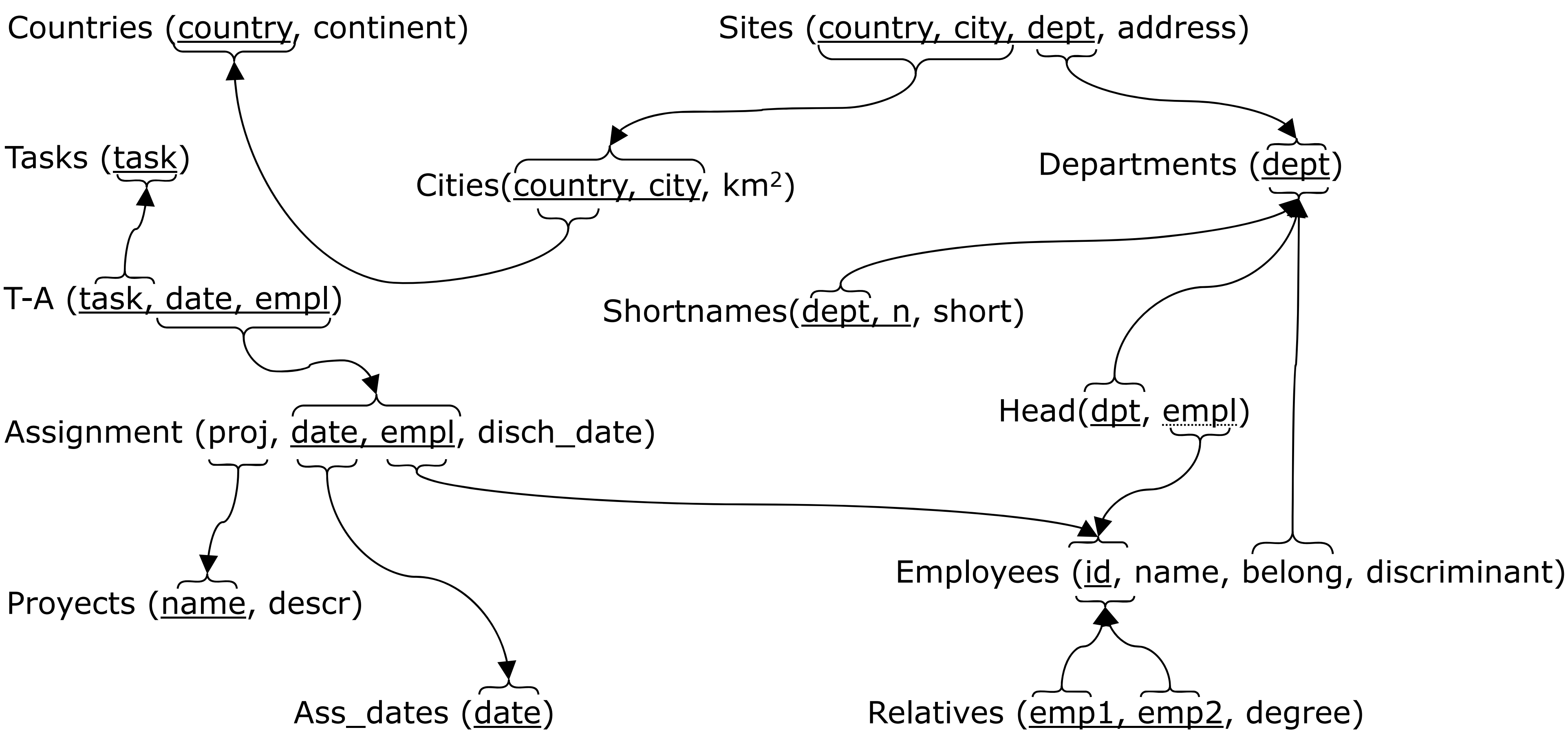
Per column	Per row
Fixed number of values	Variable number of values
Few values	Many values
Generates nulls	There are no null values
One I/O	Many I/O
Global processing	Partial processing
Natural PK	Artificial PK
Less space	More space
Hard to aggregate	Easy to aggregate
Many CHECKs	One CHECK
Lower concurrency	Higher concurrency

Example of translation

Example of conceptual schema



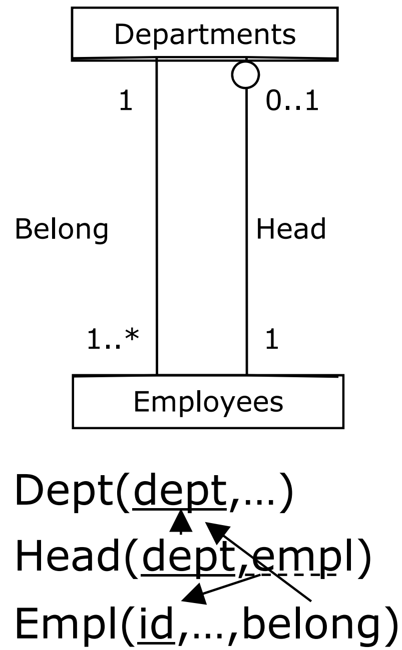
Example of logical schema



Example of constrain implementation

Constraints reflected in the conceptual schema:

1. An employee is head of zero or one department
    - Attribute “empl” in table “Head” is UNIQUE
2. Every department has exactly one head
    - Attribute “empl” in table “Head” is FK and NOT NULL
    - An assertion should be defined to check that each and every department has a head
        - We may also have implemented the association with two tables instead of three
3. An employee belongs to exactly one department
    - Attribute “belong” in table employee has been defined as FK and NOT NULL
4. Each and every department has at least one employee
    - An assertion should be defined



Closing

Summary

- Constraints
  - Deadlocks
  - Relevance of FK
- Translation of relationships
  - Binary
    - *-*
    - 1-*
    - 1-1
  - Reflexive
  - N-ary
  - Compound aggregation
  - Association classes
    - Attributes of relationships
- Multivalued attributes

Bibliography

- J. Sistac. *Sistemes de Gestió de Bases de Dades*. Editorial UOC, 2002
- J. Sistac et al. *Disseny de bases de dades*. Col·lecció Manuals, number 43. Editorial UOC, 2002
- R. Elmasri and B. Nabathe. *Fundamentals of Database Systems*. Addison-Wesley, 4th edition, 2003
- T. Teorey et al. *Database modeling and design*, 4th edition. Morgan Kaufmann Publishers, 2006