(Database) Relational Tables

Transforming an OO Model into a Relational Model

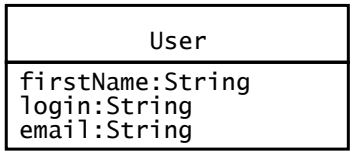
Mapping an object model to a relational database

- UML mappings
  - Each *class* is mapped to a table
  - Each class *attribute* is mapped onto a column in the table
  - An *instance* of a class represents a row in the table
  - *A many-to-many association* is mapped into its own table
  - A *one-to-many association* is implemented as buried foreign key
- Methods are not mapped

How many tables?

- Each (entity) class and each association could originate a table
  - This would generate an unnecessary large number of tables
- The goal is to optimize the number of tables generated, while guaranteeing no duplications nor attributes with undefined values

Mapping the User class to a database table



User table

id:long	firstName:text[25]	login:text[8]	email:text[32]

Primary and Foreign Keys

- Any set of attributes that could be used to uniquely identify any data record in a relational table is called a **candidate key**.
- The actual candidate key that is used in the application to identify the records is called the **primary key.**
  - The primary key of a table is a set of attributes whose values uniquely identify the data records in the table.
- A **foreign key** is an attribute (or a set of attributes) that references the primary key of another table.

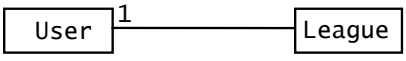
Example for Primary and Foreign Keys

User table

Primary key

firstName	login	email
“alice”	“am384”	“am384@mail.org”
“john”	“js289”	“john@mail.de”
“bob”	“bd”	“bobd@mail.ch”

Candidate key Candidate key



League table

name	login
“tictactoeNovice”	“am384”
“tictactoeExpert”	“am384”
“chessNovice”	“js289”

Attention: an attribute of User, NOT a reference to User

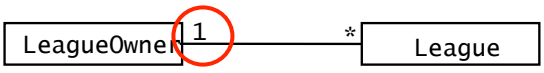
Foreign key referencing **User table**

Buried Association

- Associations with **multiplicity one** can be **implemented using a foreign key**.
  - Because the association vanishes in the table, we call this a buried association.
- For one-to-many associations we add a foreign key to the table representing the class on the “many” end.

Association one-many

- For one-to-many associations we add the foreign key to the table representing the class on the “many” end.



LeagueOwner table

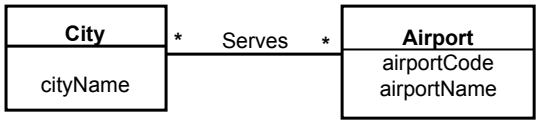
id:long	...

League table

id:long	...	leagueOwner:long

Mapping Many-To-Many Associations

What do we need in this case?



City Table

cityName
Houston
Albany
Munich
Hamburg

Airport Table

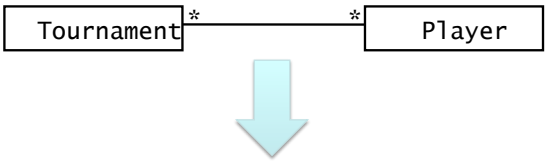
airportCode	airportName
IAH	Intercontinental
HOU	Hobby
ALB	Albany County
MUC	Munich Airport
HAM	Hamburg Airport

Serves Table

cityName	**airportCode**
Houston	IAH
Houston	HOU
Albany	ALB
Munich	MUC
Hamburg	HAM

Primary and Foreign Keys are in bold

Mapping the Tournament/Player association as a separate table



Tournament table

id	**name**	**...**
23	novice	
24	expert	

TournamentPlayerAssociation table

tournament	**player**
23	56
23	79

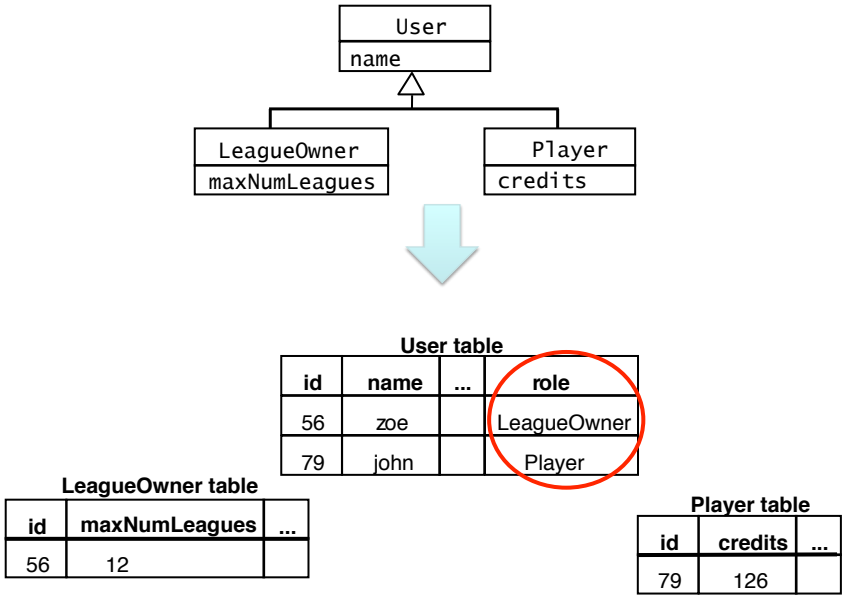
Player table

id	**name**	**...**
56	alice	
79	john	

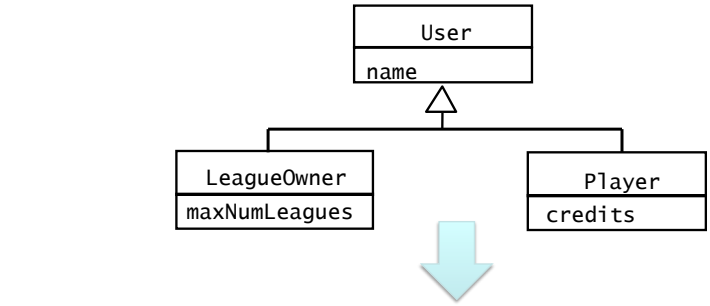
Realizing Inheritance

- Relational databases do not support inheritance
- Two possibilities to map UML inheritance relationships to a database schema
  - With a separate table (vertical mapping)
    - The superclass and the subclasses are mapped to different tables
  - By duplicating columns (horizontal mapping)
    - There is no table for the superclass
    - Each subclass is mapped to a table containing the attributes of the subclass and the attributes of the superclass

Realizing inheritance with a separate table



Realizing inheritance by duplicating columns



LeagueOwner table

id	name	maxNumLeagues	...
56	zoe	12	

Player table

id	name	credits	...
79	john	126	

Comparison: Separate Tables vs Duplicated Columns

- The trade-off is between modifiability and response time
  - How likely is a change of the superclass?
  - What are the performance requirements for queries?
- Separate table mapping
  - ☺ We can add attributes to the superclass easily by adding a column to the superclass table
  - ☹ Searching for the attributes of an object requires a join operation.
- Duplicated columns
  - ☹ Modifying the database schema is more complex and error-prone
  - ☺ Individual objects are not fragmented across a number of tables, resulting in faster queries

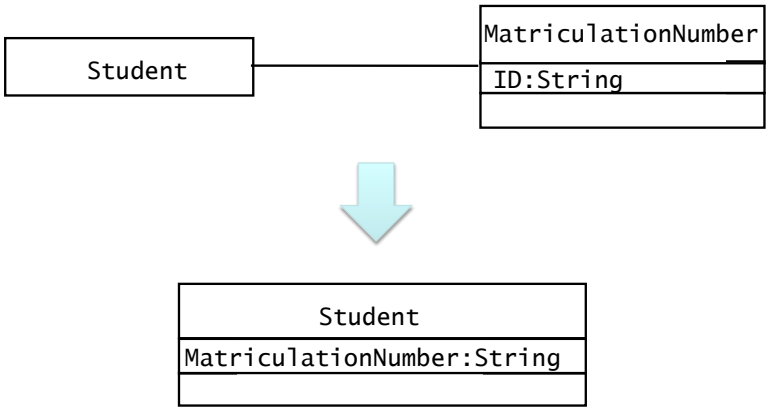
Design Optimization Activities

1. Add redundant associations:
   - What are the most frequent operations?
   - How often is the operation called? (30 times a month, every 50 milliseconds)
2. Rearrange execution order
   - Eliminate dead paths as early as possible (Use knowledge of distributions, frequency of path traversals)
   - Narrow search as soon as possible
   - Check if execution order of loop should be reversed
3. Turn classes into attributes

Implement Application domain classes

- To collapse or not collapse: attribute or association?
  - Implement entity as embedded attribute
  - Implement entity as separate class with associations to other classes
- Associations are more flexible than attributes but introduce additional nodes in a navigation (query).

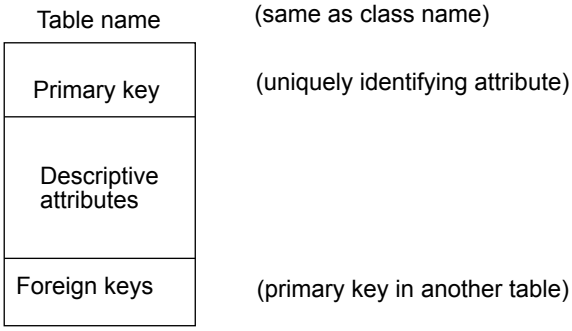
Optimization Activities: Collapsing Objects



To Collapse or not to Collapse?

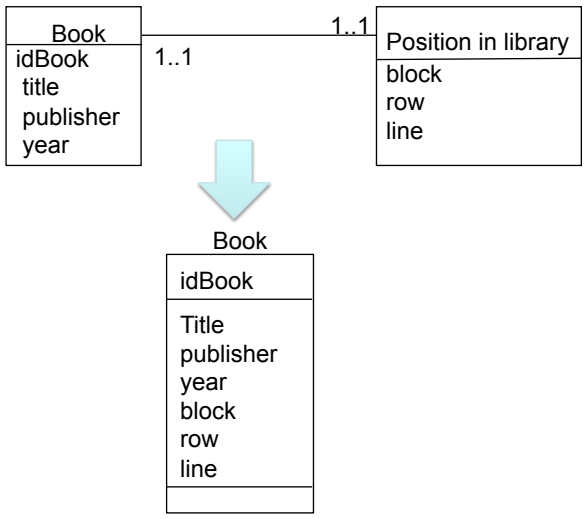
- Collapse a class into an attribute if
  - One of the classes is clearly a subordinate, and
  - the only operations defined on the attributes are Set() and Get()

A quick tour with a more compact notation



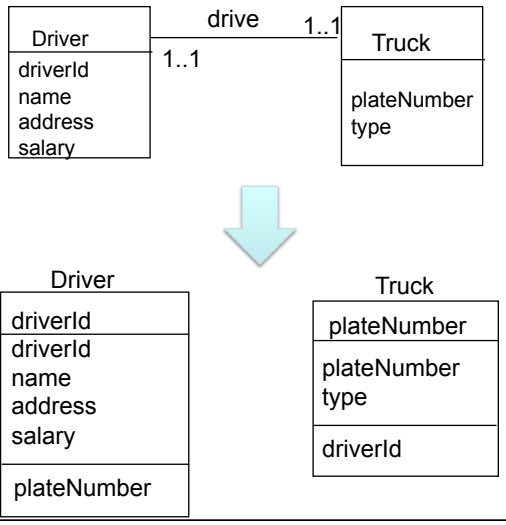
Association 1:1 (1)

CASE 1A: one-to-one; mandatory both ends



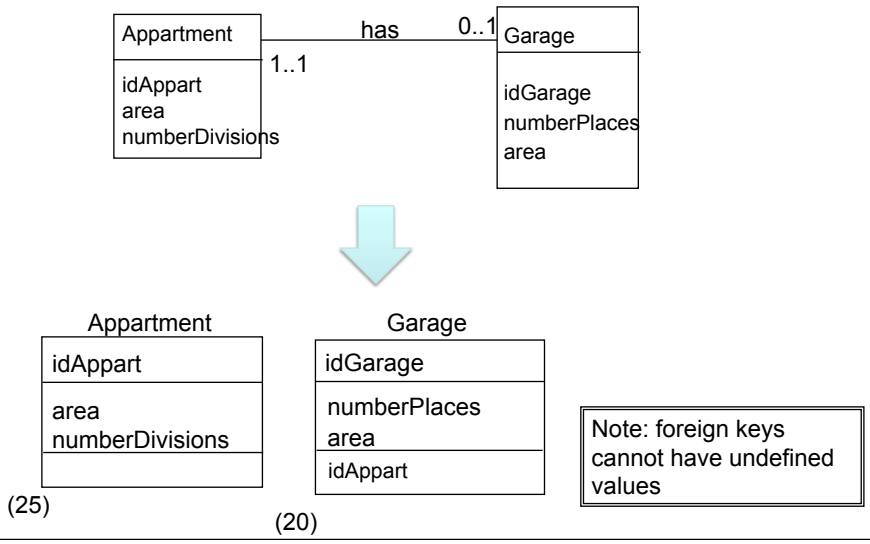
Association 1:1 (2)

CASE 1B: one-to-one; mandatory both ends



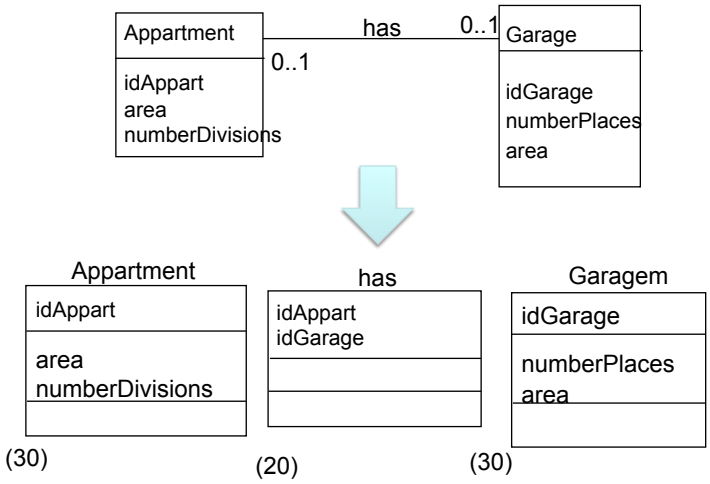
Association 1:1 (3)

CASE 1C: one-to-one; optional one end



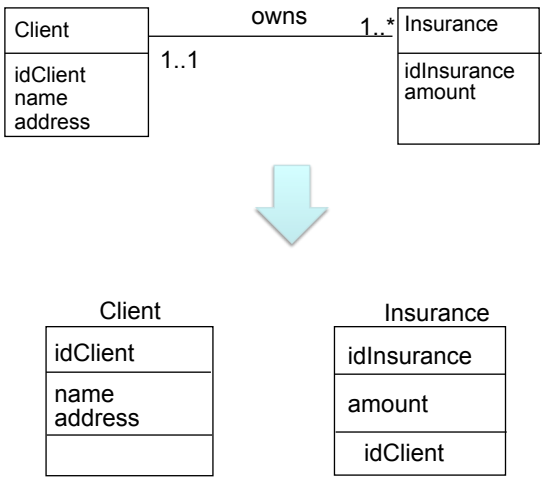
Association 1:1 (4)

CASE 1D: one-to-one; optional at both ends



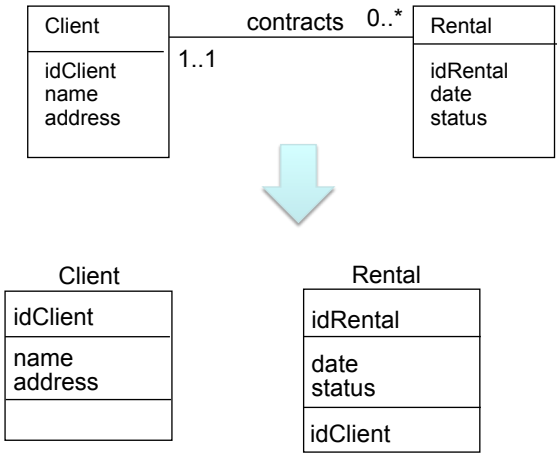
Association 1:N (1)

CASE 2A: 1:N mandatory in both directions



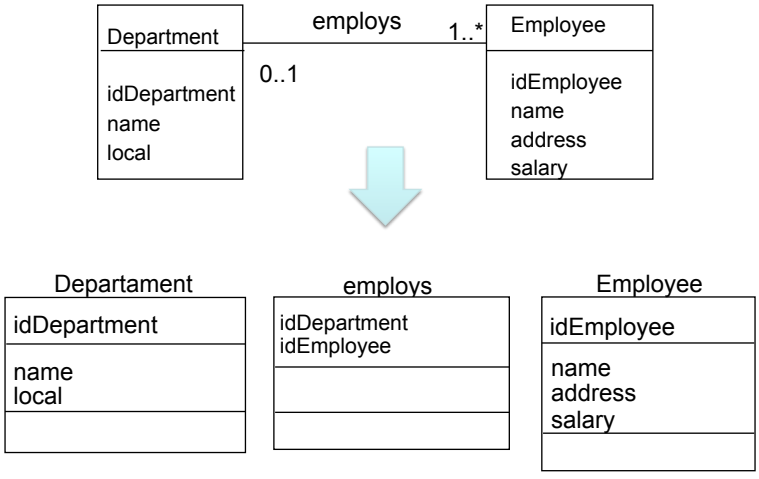
Association 1:N (2)

CASE 2B: 1:N optional in one direction



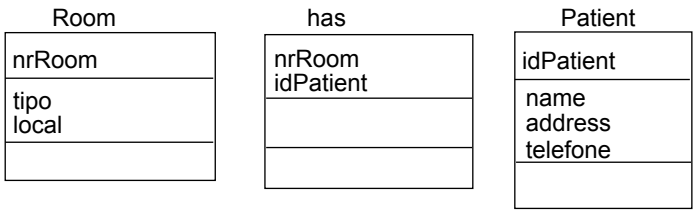
Association 1:N (3)

CASE 2C: 1:N optional in one direction



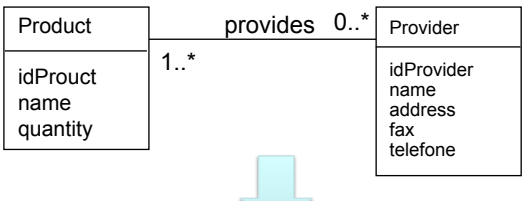
Association 1:N (4)

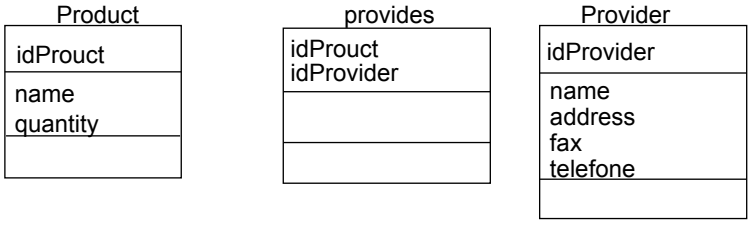
CASE 2D: 1:N optional in both directions



Association N:M

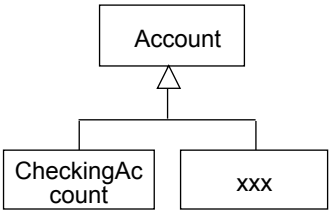
CASE 3. (independently of minimum multiplicity)





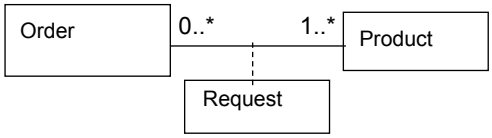
Inheritance and association class

CASE 4. Inheritance



Two or three tables, according to previous study

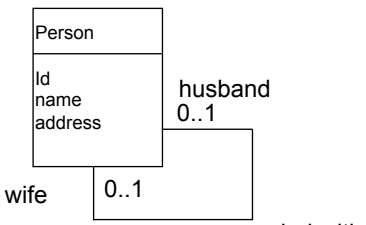
CASE 5. Association class

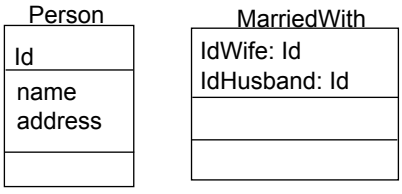


Three tables, according to previous study

Unary association (1)

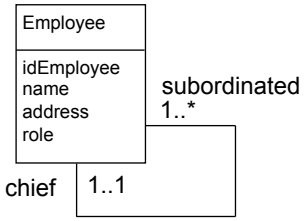
CASE 6A.

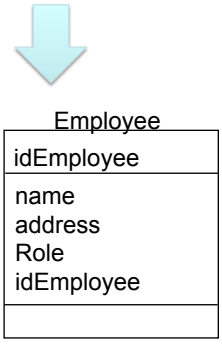




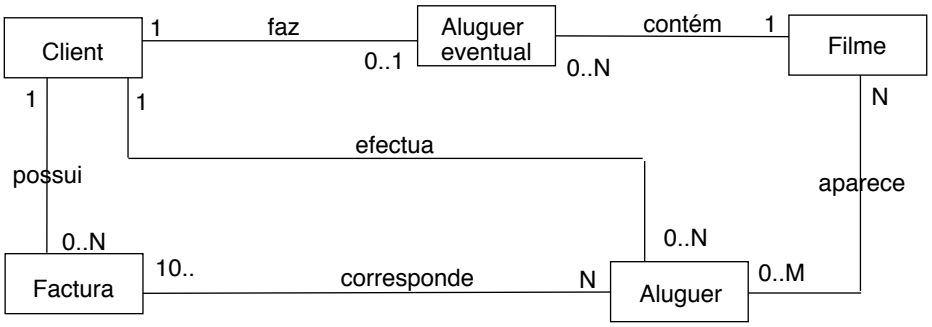
Unary association (2)

CASE 6B.

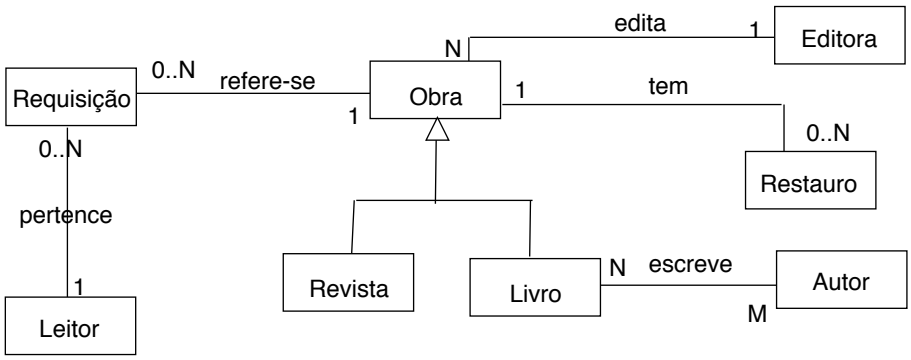




How many tables?



How many tables?



Third Normal Form

- Please review