CMSC 206: Database Management Systems

Entity Relationship and Relational models

Thomas Laurent

19/09/20120

University of the Philippines Open University

Table of contents

Introduction

Introduction

This week we will study *models* used to design databases. These models are very important as they let you *plan*/*design* your database, *foresee* potential problems (that saves you lots of headaches), and *communicate* about your DB with other stakeholders (e.g. other DB admins, the programmer using your future DB, product owners giving you requirements, ...).

Take home points

Why we use models
Difference between ER and relational model
Writing ER and relational models

Entity Relationship model

Introduction

The Entity Relationship model is used to model the world your database will describe. It is based on two elements: *entities* and *relationships*.

This model describes *the subject* of your DB, **not** your DB.

Different notation systems exist for ER diagrams, we will use a notation similar to the UML class diagrams. **Use the notation from these slides for the assignments.**
Other notations are introduced in the additional material.

Entity

An **entity** represents *an object* (physical or conceptual) in the real world with an independent existence. Similar entities are represented by an **entity type**.
e.g. professor Borromeo and I are entities of the type Person.

An entity is characterised by its **attributes**, pieces of information that can be used on their own. An attribute can have a value or no value (null value). All possible values for an attribute form its **domain**.
e.g. a person's name, a car's year of production, etc.

Person = (first_name, last_name, age, ...)

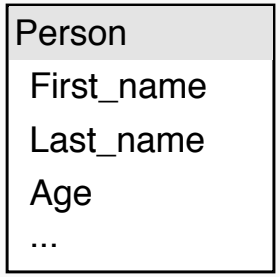


Figure 1: Textual and graphical representations of an entity.

Attribute types

Attributes can be:

- Atomic or composite

  Atomic: Not divisible, e.g. house number

  Composite: divisible into atomic attributes, e.g. address = house number + street

- Single- or multi-valued

  Single-valued: one value for each entity, e.g. birth year

  Multi-valued: one entity can have multiple values, e.g. middle name (I have 2)

- Mandatory or optional

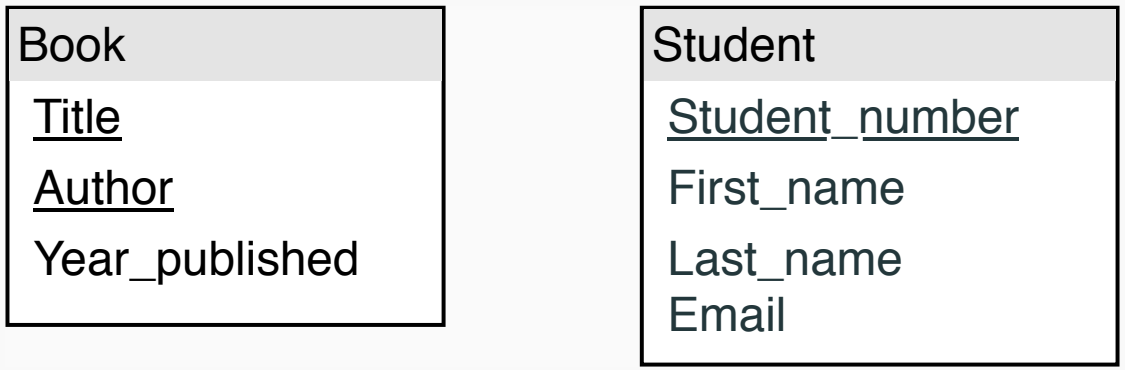
  Mandatory: each entity must have a non-null value for the attribute

  Optional: the attribute can take the null value

Primary key

The primary key of an entity type is a *minimal set of attributes* that can *uniquely identify* an entity.

The attributes making up the primary key are underlined in the diagram.



Book(Title, Author, Year_published)
Student(Student_number, First_name, Last_name, Email)

Figure 2: Primary keys example

Choosing an entity's attributes

An entity's attribute must:

- be directly linked to this entity
- not be influenced by time (should use birth date, not age)
- not be influenced by other entities (number of siblings not an attribute if we can count the siblings from the DB)
- avoid data redundancy

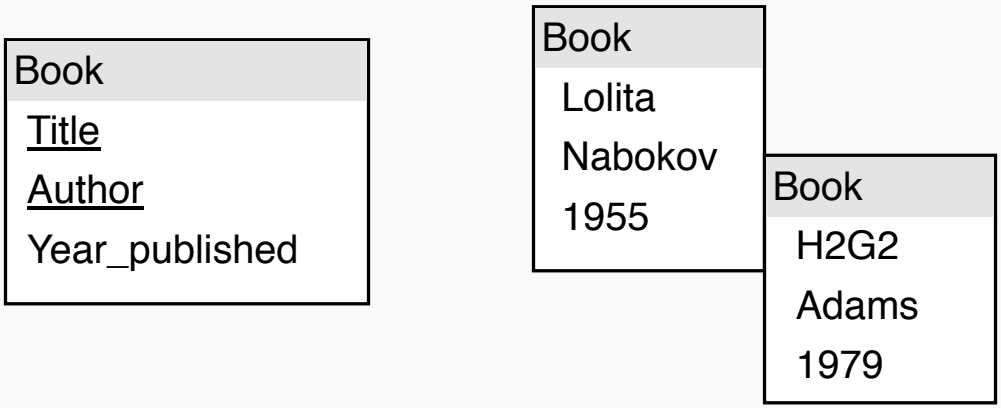


Figure 3: Entity type and entities

Relationship

A relationship represents a *semantic link between entities* that is of interest to the information system. The number of entities involved in a relationship is its **degree**. Relationships form relationship types between entity types.
e.g. Relationship type Order between entity types Client and Product

A relationship can be characterised by *attributes* in the same way as an entity.
e.g. date and quantity in the Ordered relationship type.

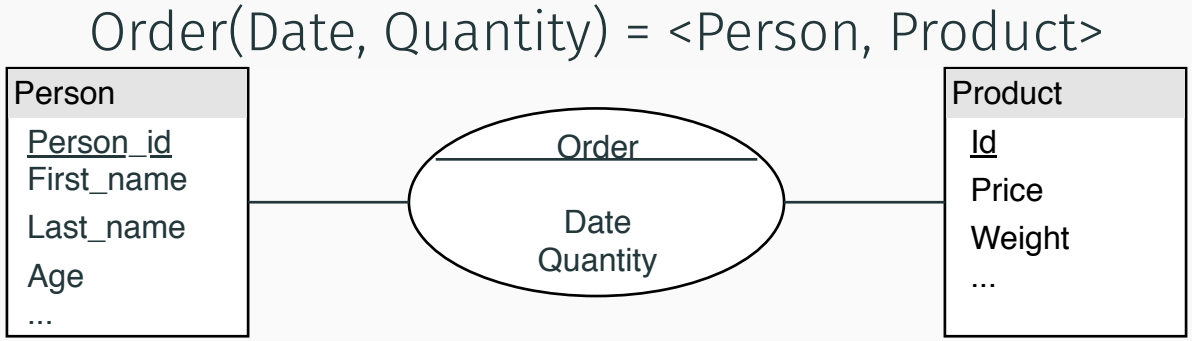


Figure 4: Textual and graphical representations of a relationship.

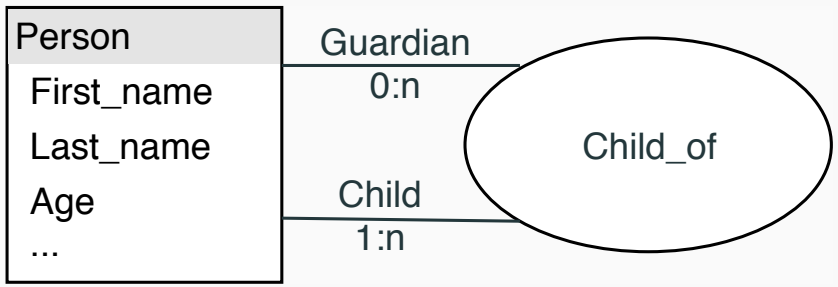
Role names and constraints

Recursive relationships are established between entities of the same type. In this case we want to use labels to identify the role of each entity in the relationship. See the example below.

Constraints can be put on the number of relationships an entity can be a part of:

Cardinality ratio: maximum number of relationships an entity can be a part of. 1 or n

Participation constraint: minimum number of relationships an entity can be a part of. 0 or 1



A guardian can have no child or up to n children.
A child must have one or up to n guardians

Figure 5: Child_of relationship with role names and constraints

Relational model

Introduction

The relational model is a *formal, logical* data model. It was proposed by Codd in 1970 [1] and offers very strong theoretical backing through the relational algebra (week 5). It is also at the base of the normalisation theory we will see next week.

The relational model is at the base of relational DBMSes (postgreSQL, mySQL, SQL server, ...) and is *implemented* through SQL.

The relational model represents the database as a collection of *relations* (not relationships!).

Relations

Informally, relations can be viewed as a *flat file*, or table, representing the data and identified by a *name*.

The ”column headers” of a relation are called *attributes*. The number of attributes in a relation is called its degree, and each attribute is characterised by its *domain*, the set of possible values for the attribute. These values are *atomic*, i.e non dividable.

A ”row” of the relation, i.e a set of (attribute/value) pairs, is called a *tuple*, or *n-tuple*.

A relation r of degree n is thus a (non ordered) set of n-tuples, $r = \{t_1, t_2, ..., t_m\}$. The relation is characterised by its name and attributes which form its *schema*: $R(A_1, A_2, ..., A_n)$.

Relation example

Relation Name

STUDENT

Attributes

Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Schema: Student(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

Figure 6: Example of a relation, from Fundamentals of Database Systems

SuperKey, Candidate Key and Primary Key

A *superkey* is any set of attributes $\{A_i\}_{i \in \{0..n\}}$ of a relation where no two tuples can have the same set of values for these attributes. i.e., a superkey is any set of attributes that uniquely identifies tuples.

A *candidate key* is a minimal superkey, i.e. any superkey so that there is no smaller superkey.

The *primary key* of a relation is a particular candidate key used to identify tuples. The primary key is represented by underlining its attributes. **Every relation must have a primary key**

Foreign key

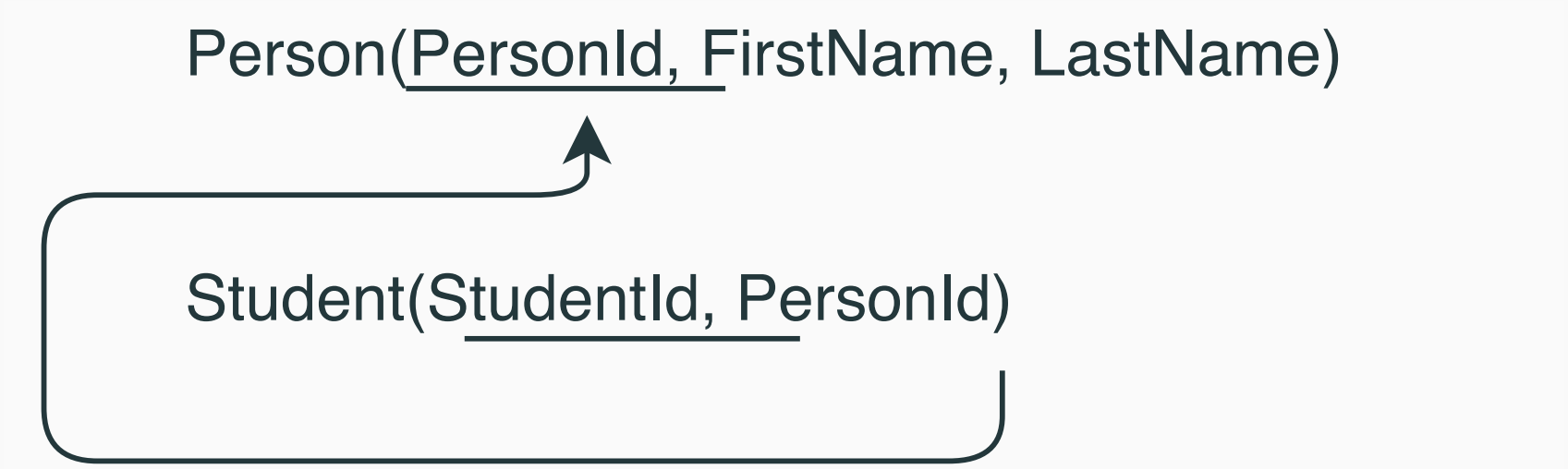
Superkeys, candidate keys and primary keys are related to a *single relation's schema*. A *foreign key* forms a constraint between *two relations*.

Let $R_1 = \{A_1, A_2, ..., A_n\}$ and $R_2 = \{B_1, B_2, ..., B_m\}$ be two relations. Let $\{A_i\}$ be the primary key for R_1. $\{B_i\}$ is a *foreign key* that *references* R_1 iff:

- $\{B_i\}$ and $\{A_i\}$ have the same domain
- For every tuple t_2 of R_2, the value of $\{B_i\}$ is NULL or occurs in a tuple t_1 of R_1

A foreign key is represented by an arrow from the referencing attribute(s) to the referenced attribute(s).

Foreign key illustration



A student is a person so must have an existing PersonId. We create a foreign key in the Student relation that references the Person relation.

Figure 7: Example of Foreign Key

Integrity constraint

A *database schema* is the sum of the the relation schemas and of *integrity constraints*, rules to make sure the database is in a coherent state. The different kinds of constraints are:

entity integrity constraint: specifies that a primary key's value can not be NULL.

domain integrity constraint: specifies that an attribute's value must respect its domain

referential integrity constraint: enforces the foreign key mechanism

other constraints: other constraints can be specified in the DB using mechanisms such as *triggers* or *assertions*.

From ER model to Relational model

From logical to physical model

When designing a DB, we first draft a conceptual model, with the ER model, before designing the physical model, with the relation model.

This process echoes the levels of abstractions we saw last week. We first design the conceptual model with the end user/ stakeholders, who do not really care how the DB is implemented but who know what information they need. We then make a logical relational model and finally a physical relational model by adding the domains of the attributes.

We will now see how to transform our ER model into a relational model with three simple rules.

Rule 1

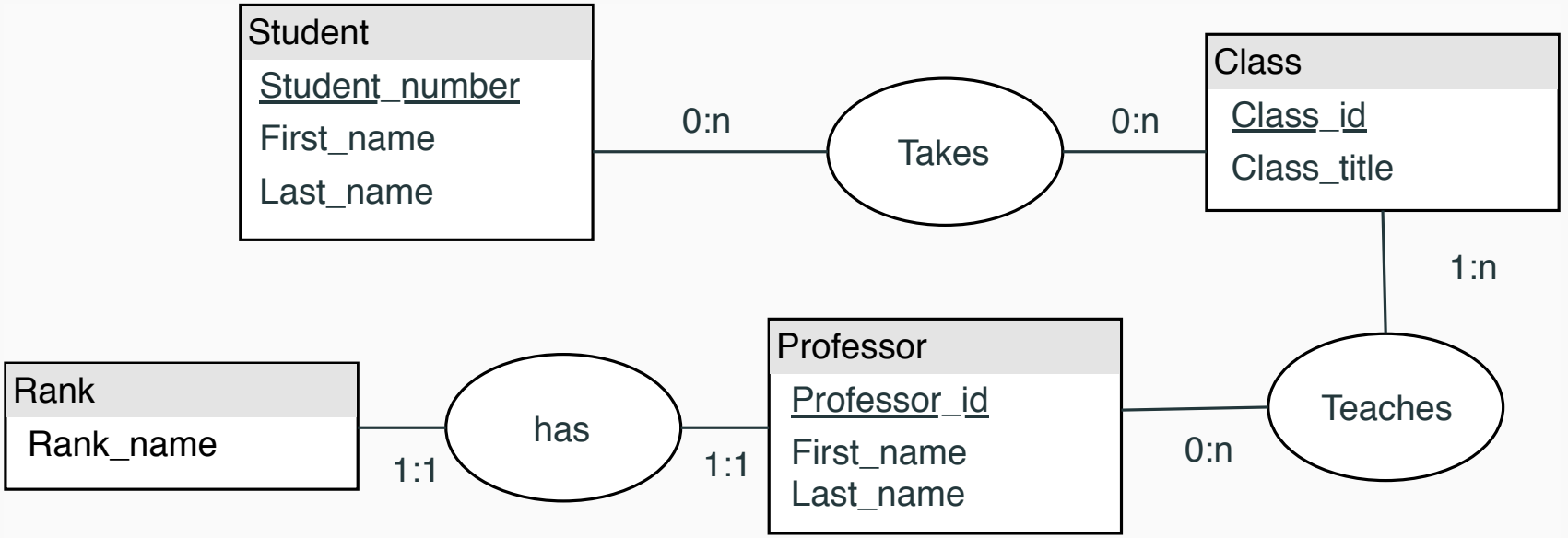
Every entity becomes a relation.

The attributes of the entity become the relation’s attributes.

The primary key of the entity is the relation’s primary key.

Rule 1 - illustration

Let's illustrate the rules with an example.



By applying rule 1 to this ER diagram we obtain the following relational schema:

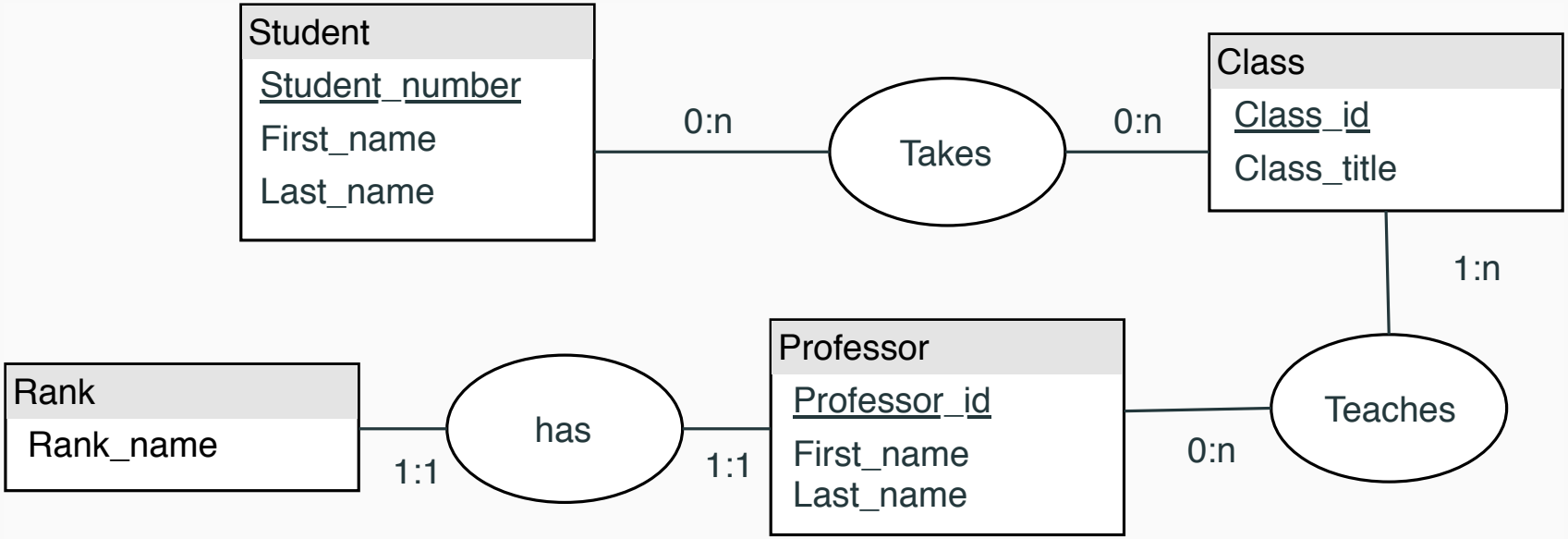
Student(Student_number, First_name, Last_name)
Class(Class_id, Class_title)
Professor(Professor_id, First_name, Last_name)
Rank(Rank_name)

Rule 2

Every binary relationship with a cardinality ratio of 1 (0:1 or 1:1 relationship) is translated as a foreign key in the relation having the cardinality ratio of 1 and referencing the other relation.

The attributes of the relationship are added to the relation alongside the foreign key.

Rule 2 - illustration



By applying rule 1 and 2 to this ER diagram we obtain the following relational schema:

Student(Student_number, First_name, Last_name)
Class(Class_id, Class_title)
Professor(Professor_id, First_name, Last_name, Rank_name)
    Rank_name -> Rank
Rank(Rank_name)

Rule 3

Remaining relationships become relations. The primary key of these relations are the primary keys of entities included in the relationship.

A foreign key is created from the new relation to all the relations included in the relationship.

The attributes of the relationship become attributes in the new relation.

Rule 3 - illustration

By applying all three rules to this ER diagram we obtain the following relational schema:

Student(Student_number, First_name, Last_name)
Class(Class_id, Class_title)
Professor(Professor_id, First_name, Last_name, Rank_name)
    Rank_name -> Rank
Rank(Rank_name)
Takes(Student_number, Class_id)
    Student_number -> Student, Class_id -> Class
Teaches(Class_id, Professor_id)
    Class_id -> Class, Professor_id -> Professor

Logical relational model

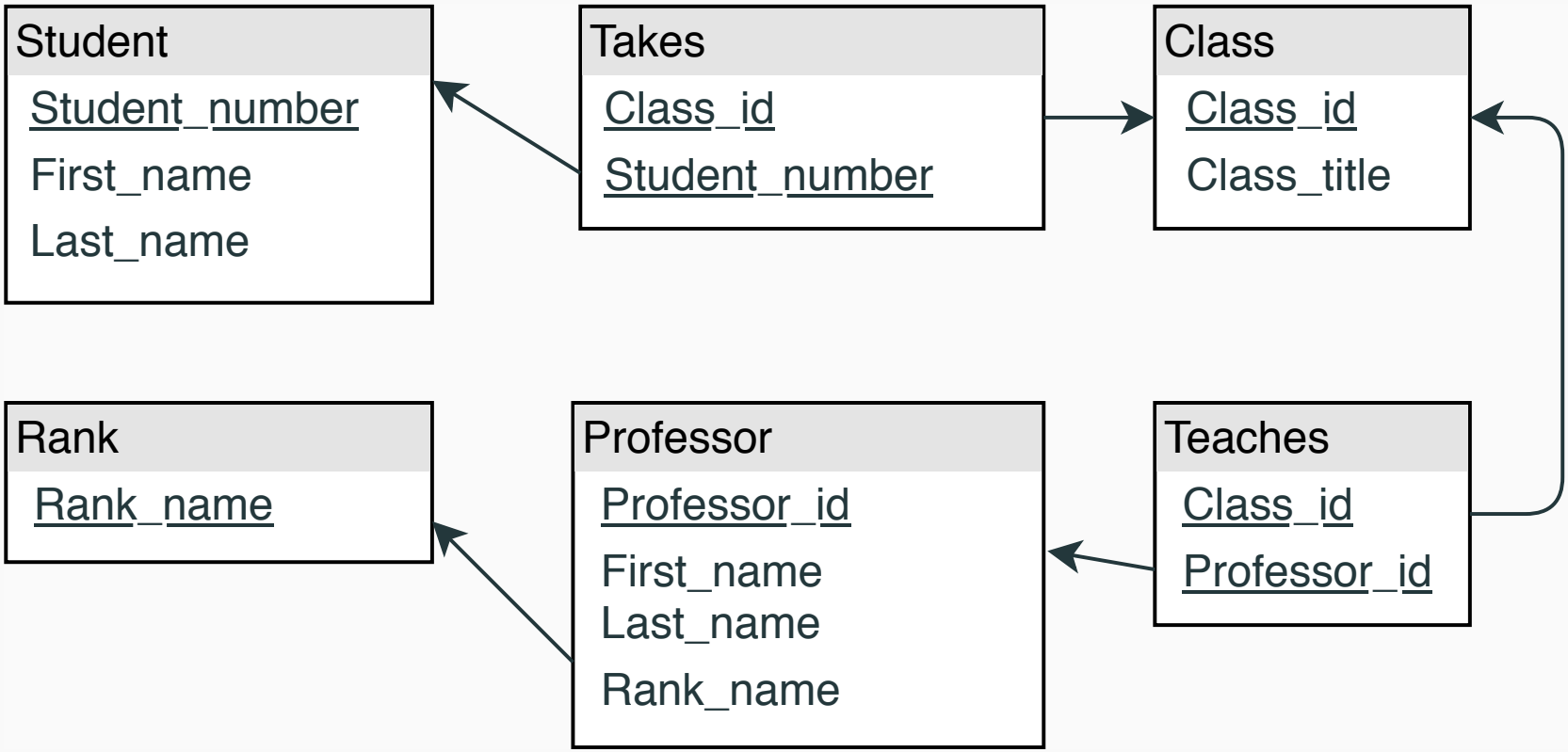


Figure 8: Logical relational model for the illustration

Physical relational model

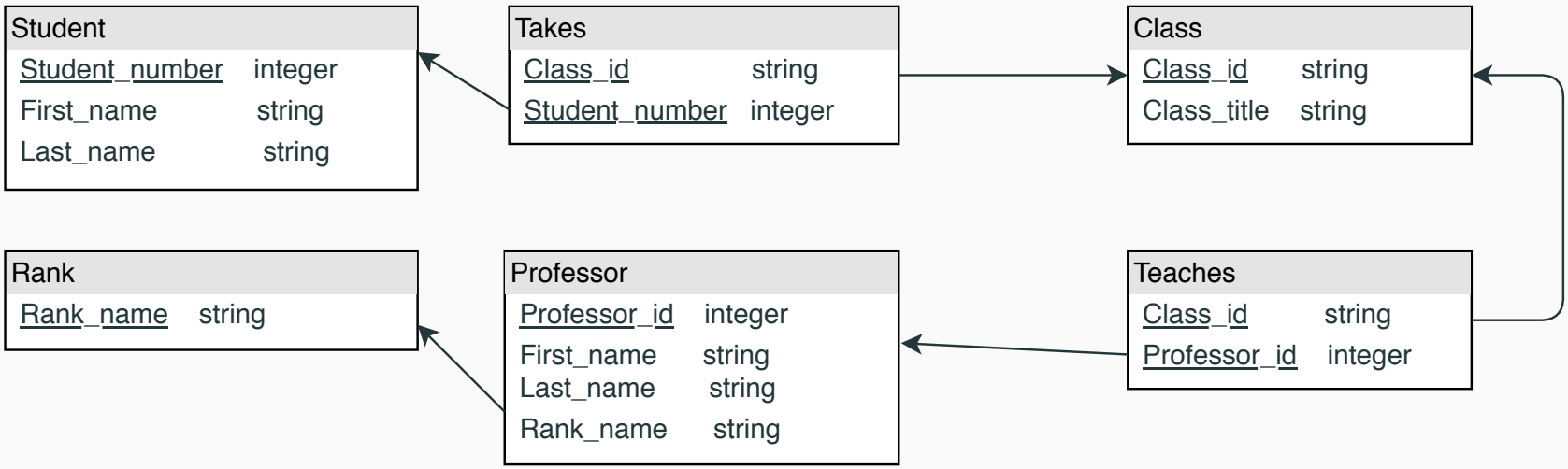


Figure 9: Physical relational model for the illustration

Thank you, see you next week!

References i

E. F. Codd.
A relational model of data for large shared data banks.
Communications of the ACM, 13(6):377–387, 1970.