Entity Relationship and Relational Models

CMSC 206 – Database Management Systems

Revision History
05 Oct 2022 : Template update – Ria Borromeo
19 Sep 2019 : Main content – Thomas Laurent

Outline

1. Introduction
2. Entity Relationship Model
    - Entities
    - Relationships
3. Relational Model
    - Definition
    - Keys and Constraints
4. From ER Model to Relational Model
5. Appendix

Introduction

In this lecture, 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 can save you lots of headaches), and ***communicate*** about your DB with other stakeholders such as other DB admins, the programmer who will use your DB, and product owners who give you the requirements.

Points to Remember

- Why we use models
- Difference between ER and Relational modesl
- How to create ER and Relational models

Entity Relationship Model

Introduction

- The Entity-Relationship model is used to model the world that 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 UML class diagrams. Please use the notation from these slides for the assignments.
- Other notations are introduced in the additional materials.

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. Dean Maranan and I are entities of the type Person.
- An entity is characterized by its **attributes**, which are pieces of information that can be used on their own. An attribute can have a value or no value (null). All possible values for an attribute form its **domain**.
  - E.g. a person's name, a car's year of production, etc.

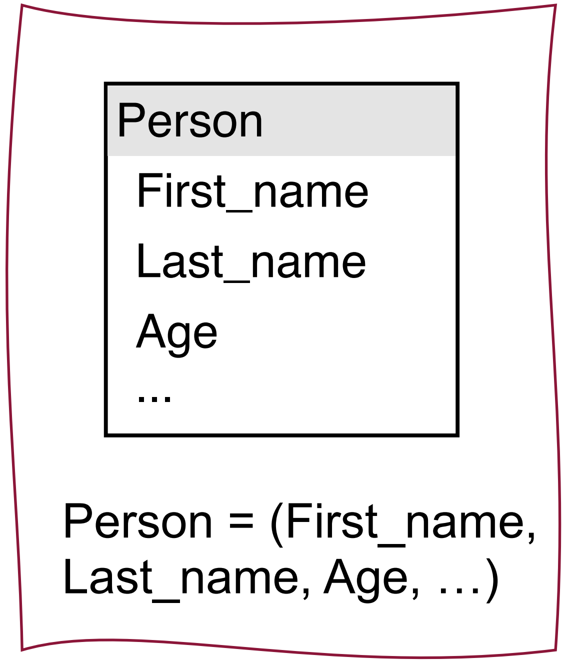


Figure 1: Textual and graphical representations of an entity

Review of Definitions

- **Entity** – represents an object in the real world with an independent existence
- **Attributes** – pieces of information that can be used on their own
- **Domain** – all possible values for an attribute

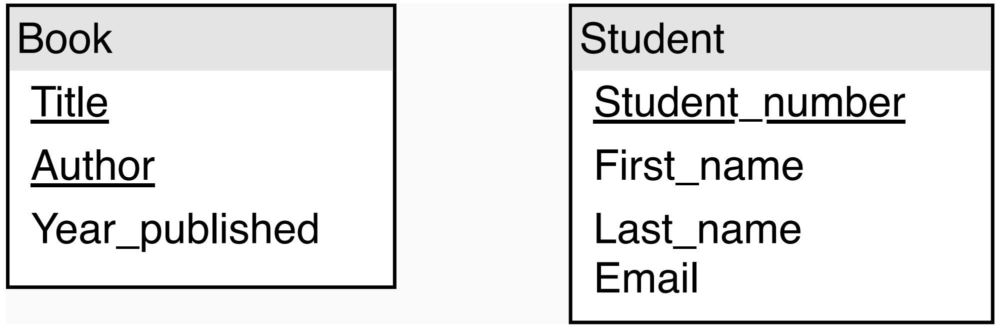
Attribute Types

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-valued or Multi-valued**
  - **Single-valued**: each entity has one value, e.g. birth year
  - **Multi-valued**: one entity can have multiple values, e.g. contact nos.
- **Mandatory or Optional**
  - **Mandatory**: each entity must have a non-null value for the attribute
  - **Optional**: the attribute can take the non-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 the entity
- not influenced by time (for example, instead of storing age, use birth date)
- not be influenced by other entities (for example, the number of siblings must not be an attribute if we can count it from the DB)
- avoid data redundancy

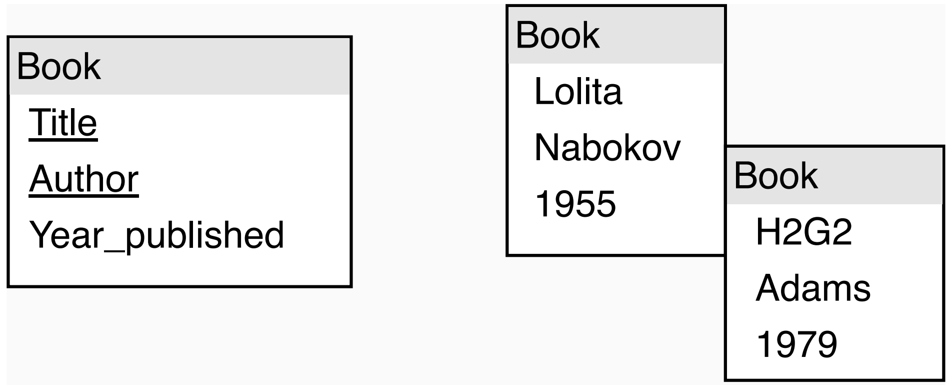


Figure 3. Entity type and entities

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 quanity in the Order relationship type.

Order(Date, Quantity) = <Person, Product>

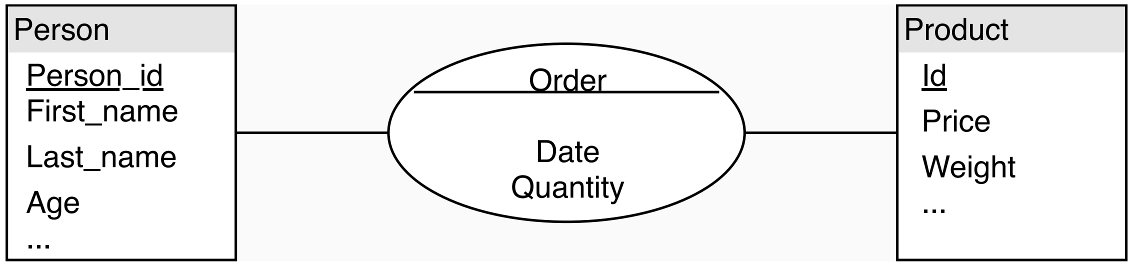


Figure 4. Textual and graphical representations of a relationship

Role names and constraints

Recursive relationships are established between entities of the same type. In 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 part of:

Cardinality ratio – maximum number of relationships that 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

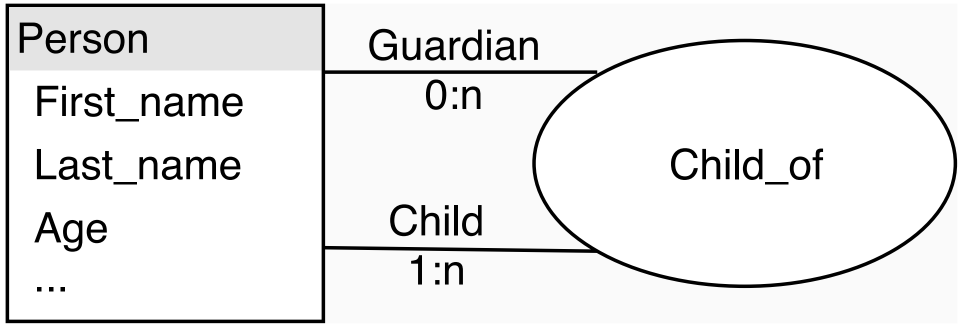


Figure 5. Child_of relationship with role names and constraints

In Figure 5, we can see that a guardian can have no child or up to n children. A child must have one or up to n guardians.

Relational Model

The Relational Model

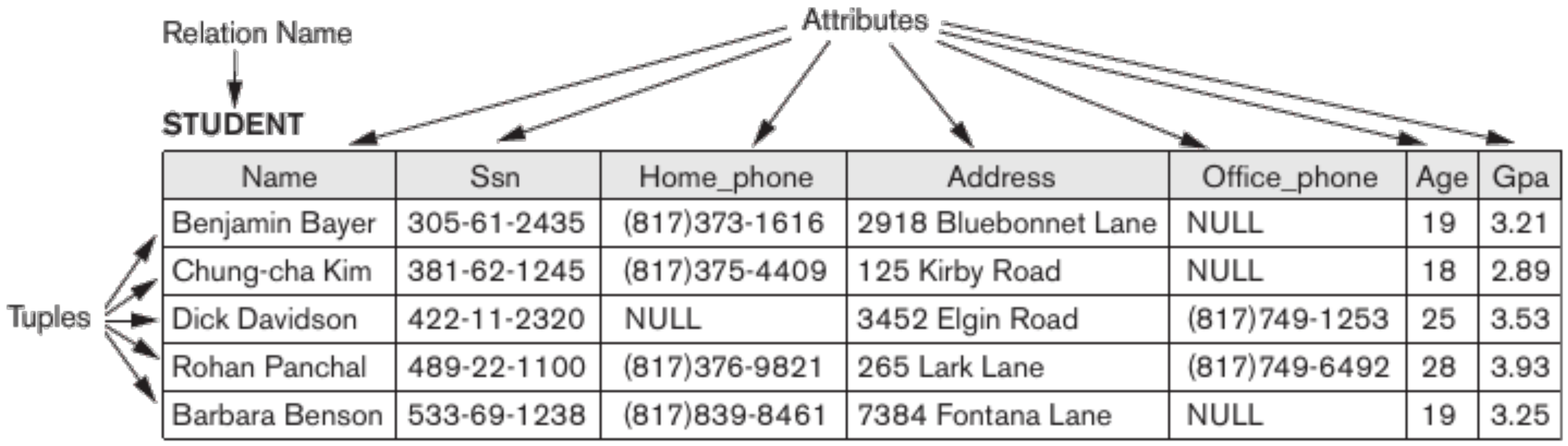
- The relational model is a ***formal***, ***logical*** data model. It was proposed by Codd in 1970[1] and offers very strong theoretical backing through the relational algebra.
- The relational mode is the base of relational DBMSes (PostgreSQL, MySQL, SQL Server, etc.) and is implemented through SQL. It is also the base of the normalization theory.
- The relational model represents the database as a collection of relations (not relationships!)

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

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 characterized by its ***domain***, the set of possible values for the attribute. The values are ***atomic***, which means they cannot be divided.
- A row in a 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, $r = \{t_1, t_2, \ldots, t_m\}$. The relation is characterized by its name and attributes, which form its schema: $R(A_1, A_2, \ldots, A_n)$.

Relation: Example



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 attriutes $\{A_i\}_{i \in \{0..n\}}$ 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.

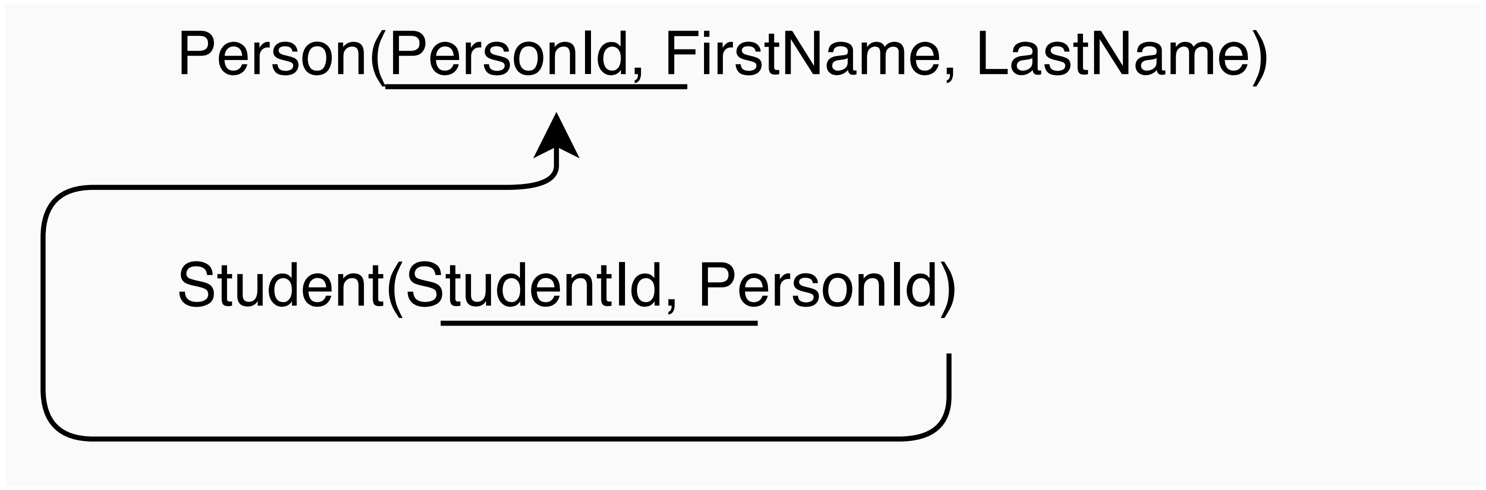
Let $R_1 = \{A_1, A_2, ..., A_n\}$ and $R_2 = \{B_1, B_2, ..., B_m\}$ be two relations. Let {Ai} be the primary key for R_1. $\{B_i\}$ is a foreign key that references R_1 iff:

- $\{B_i\}$ 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


Person(PersonId, FirstName, LastName)
Student(StudentId, PersonId)

A student is a person so it 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 relation schemas and of ***integrity constraints*** or the 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 cannot 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 mode, with the ER model, before designing the physical model with the relational model.

This process echoes the levels of abstractions in the previous topic

1. Design the conceptual model with the end user / stakeholders, who know what information they need but do not really care about how the DB is implemented.
2. Make a logical relational model
3. Design 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 (3) 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.

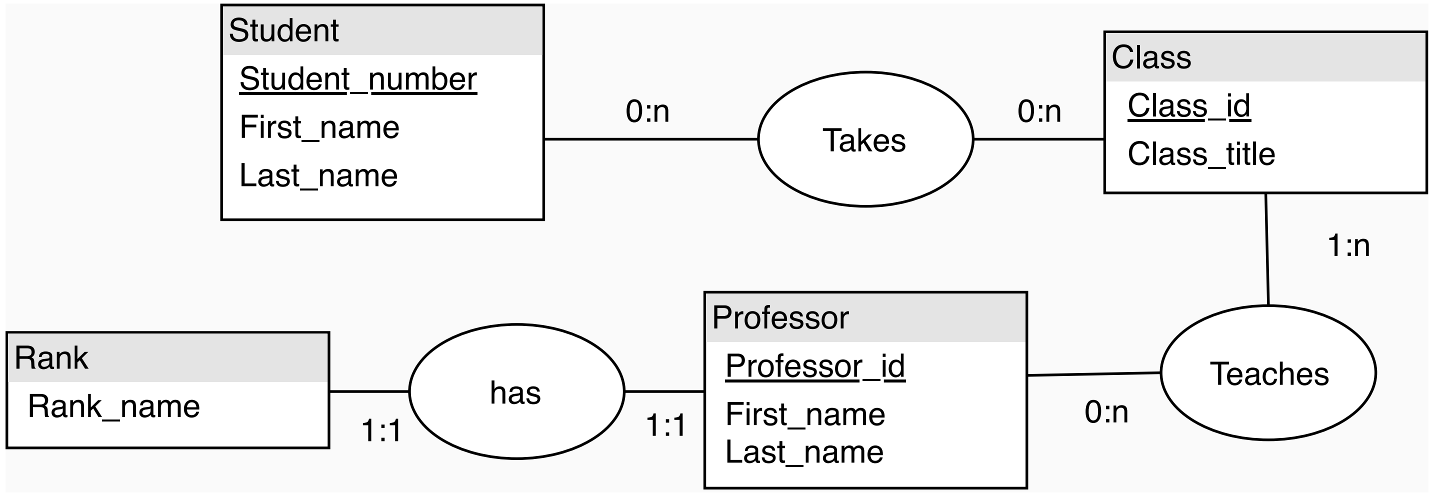


Figure 8. ER Model for the Illustration

By applying Rule 1 to Figure 8, we obtain the ff relational schema:

1. **Student**(<u>Student_number</u>, First_name, Last_name)
2. **Class**(<u>Class_id</u>, Class_title)
3. **Professor**(<u>Professor_id</u>, First_name, Last_name)
4. **Rank**(Rank_name)

Rule 2

Every binary relationship with a cardinality ration of 1 (0:1 or 1:1 relationship) is translated as a foreign key in the relation.

- The cardinality ratio is 1
- It must reference the other relation
- The attributes of the relationship are added to the relation alongside the foreign key.

Rule 2 - Illustration

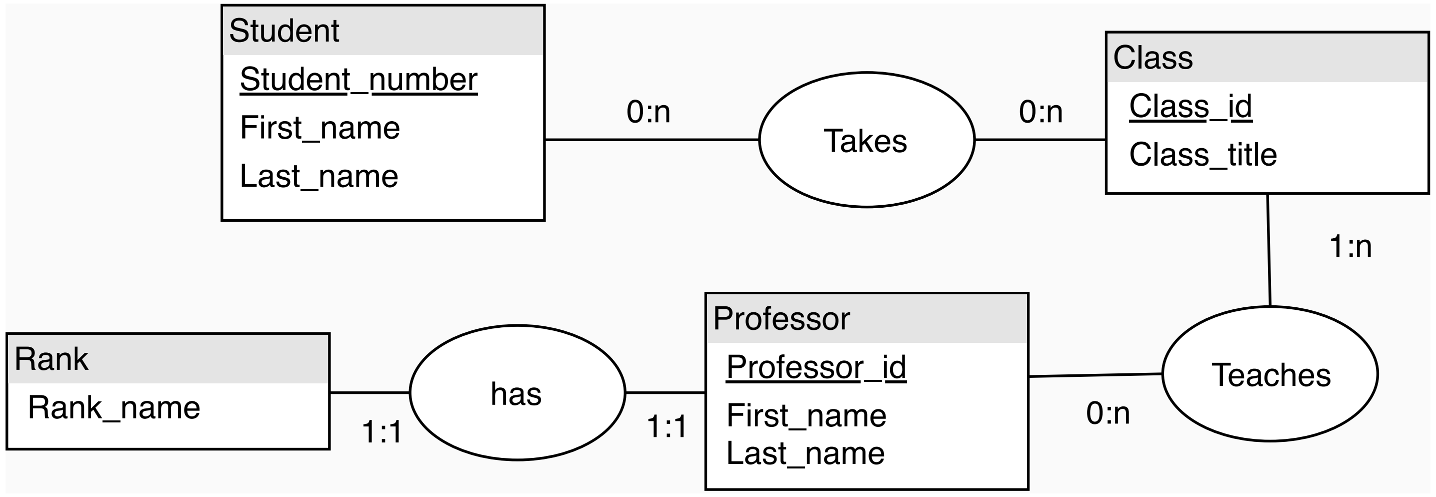


Figure 8. ER Model for the Illustration

By applying Rule 1 and Rule 2 to Figure 8, we obtain the ff relational schema:

1. **Student**(<u>Student_number</u>, First_name, Last_name)
2. **Class**(<u>Class_id</u>, Class_title)
3. **Professor**(<u>Professor_id</u>, First_name, Last_name, Rank_name)
   - *Rank_name → Rank*
4. **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 Figure 8, we obtain the following relational schema:

1. **Student**(Student_number, First_name, Last_name)
2. **Class**(Class_id, Class_title)
3. **Professor**(Professor_id, First_name, Last_name, Rank_name)
   - Rank_name -> Rank Rank(Rank_name)
4. **Takes**(Student_number, Class_id)
   - Student_number -> Student, Class_id -> Class
5. **Teaches**(Class_id, Professor_id)
   - Class_id -> Class, Professor_id -> Professor

Logical Relational Model

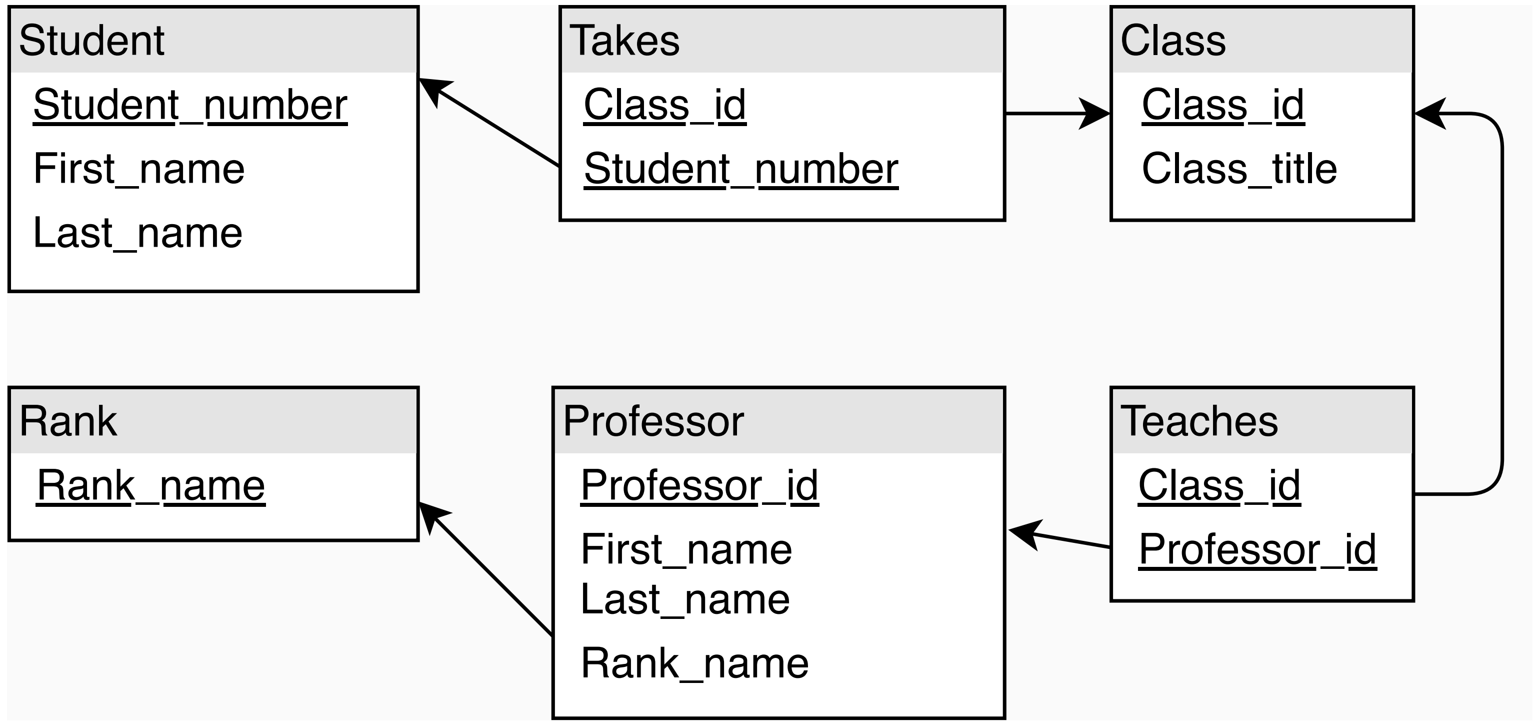


Figure 9. The resulting Logical Relational Model of the ERD in Figure 8

Physical Relational Model

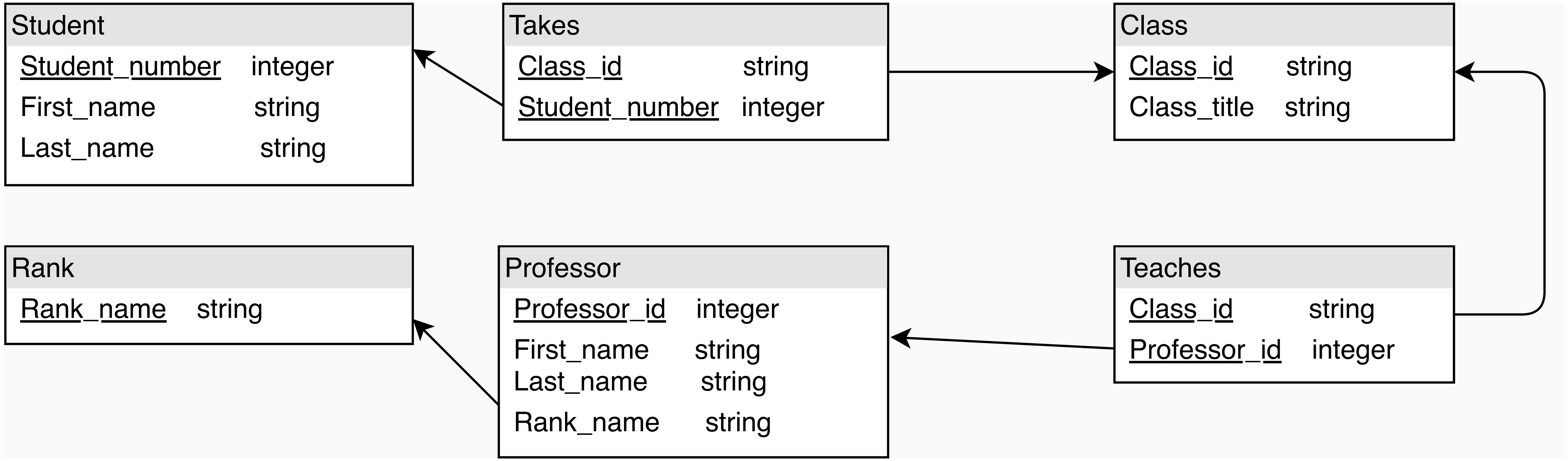


Figure 9. The resulting Physical Relational Model of the ERD in Figure 8