Class diagram and schema design

Key terms

Class diagram

> A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Class diagram

A class diagram is a graphical representation of a system's low level implementation and describes the realtionships between the various components of the system. A class diagram represents the following components of a system:

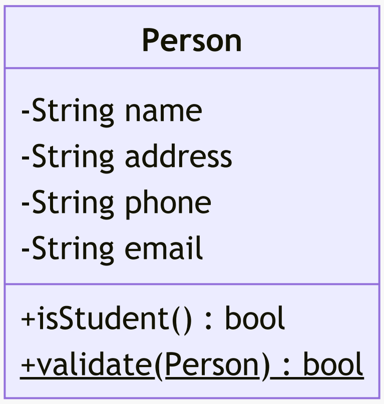
- Classes
- Interfaces
- Abstract classes

The following data points about the relationships between the components are also represented:

- Which classes implement which interfaces?
- Which class is a subclass?
- Which class is a superclass?
- Which class is an attribute of another class?

Entities and attributes

A class is represented by a rectangle with the name of the class in the middle. The attributes and methods of the class are presented in the rectangle. The attributes are presented in the top half of the rectangle and the methods are presented in the bottom half of the rectangle.



Attributes

Attributes are the properties of a class. An attribute is represented in the form of

> [access modifier] [attribute name]: [attribute type] The following are parts of an attribute:

- `Access modifier` - Visibility of the attribute. It can be one of the following:
  - `+` public
  - `-` private
  - `#` protected
  - `~` package
- `Attribute name` - Name of the attribute
- `Attribute type` - Type of the attribute. It can be a primitive type or a class type.

Methods

Methods are the operations that can be performed on a class. A method is represented in the form of

> [access modifier] [method name]([parameter list]): [return type]

The following are parts of a method:

- `Access modifier` - Visibility of the method. Same as the access modifier of an attribute.
- `Method name` - Name of the method
- `Parameter list` - List of parameters that the method takes but instead of the parameter name, the parameter type is used. The parameters are separated by commas.

- `Return type` - Type of the value returned by the method. It can be a primitive type or a class type.

Static methods and attributes are underlined.

Interfaces and abstract classes

An interface is a contract that a class must implement. An interface is represented by a rectangle with the name of the interface in the middle surrounded by an angled bracket. The methods of the interface are presented in the rectangle.

Similarly, abstract classes and methods are italicized.

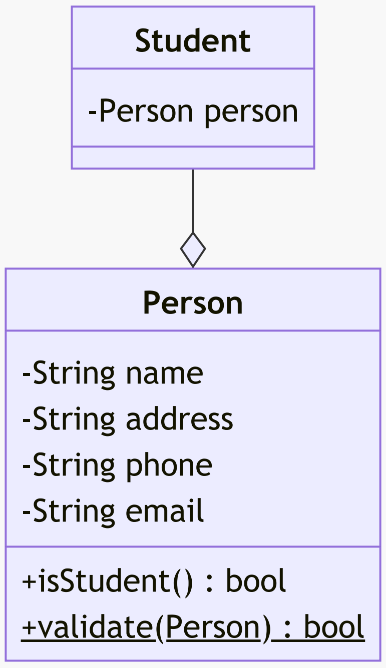
Relationships

There are 2 major types of relationships between classes:

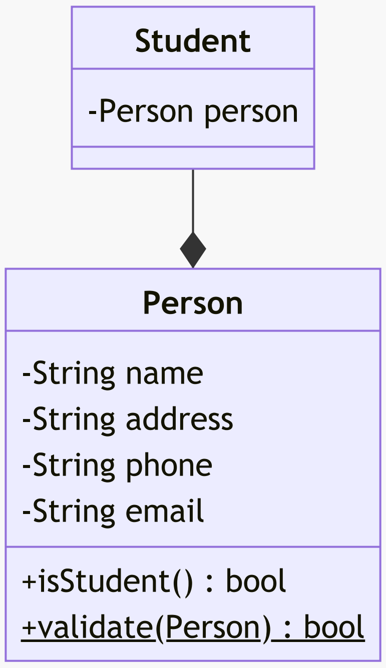
- `Inheritance` - A relationship between a superclass and a subclass. The subclass inherits the attributes and methods of the superclass.
- `Association` - A relationship between 2 classes. The classes are associated with each other. The association can be one of the following:
  - `Aggregation` - A relationship between 2 classes where the lifetime of the child class is dependent on the lifetime of the parent class. The child class can exist without the parent class.
  - `Composition` - A relationship between 2 classes where the lifetime of the child class is dependent on the lifetime of the parent class. The child class cannot exist without the parent class.

Inheritance is represented by a line with an arrow pointing to the superclass. Sometimes the arrow is replaced by a triangle.

Aggregation is represented by a line with a diamond at the end of the line.



Composition is represented by a line with a filled diamond at the end of the line.



Cardinality

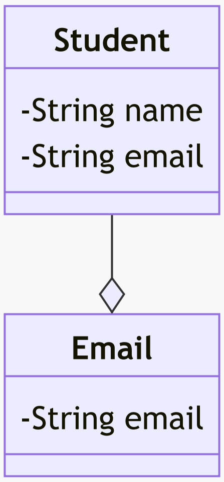
> Cardinality is the maximum times an entity can relate to an instance with another entity or entity set.

> the number of interactions entities have with each other.

One to One (1:1)

> A "one-to-one" relationship is seen when one instance of entity 1 is related to only one instance of entity 2 and vice-versa

A student can only have one email address and one email address can be associated with only one student.

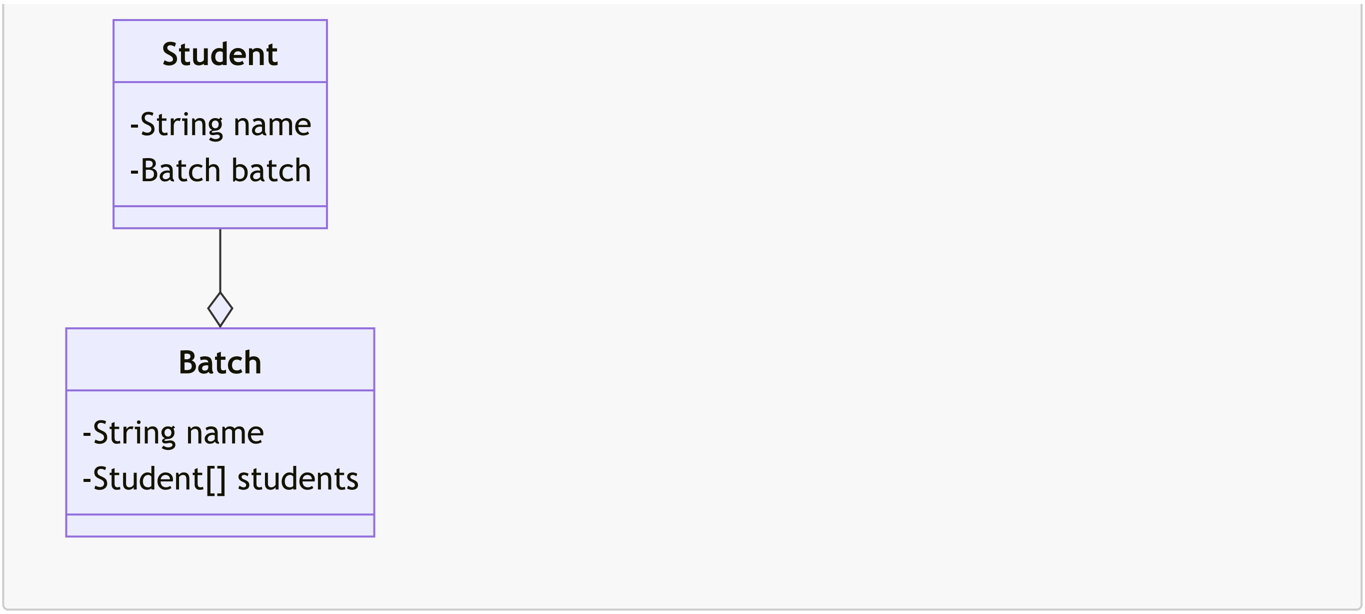


An attribute shared by both entities can be added to either of the entities.

One to Many or Many to one (1:m or m:1)

> When one instance of entity 1 is related to more than one instance of entity 2, the relationship is referred to as "one-to-many.

A student can only be associated with one batch, but a batch can have many students.

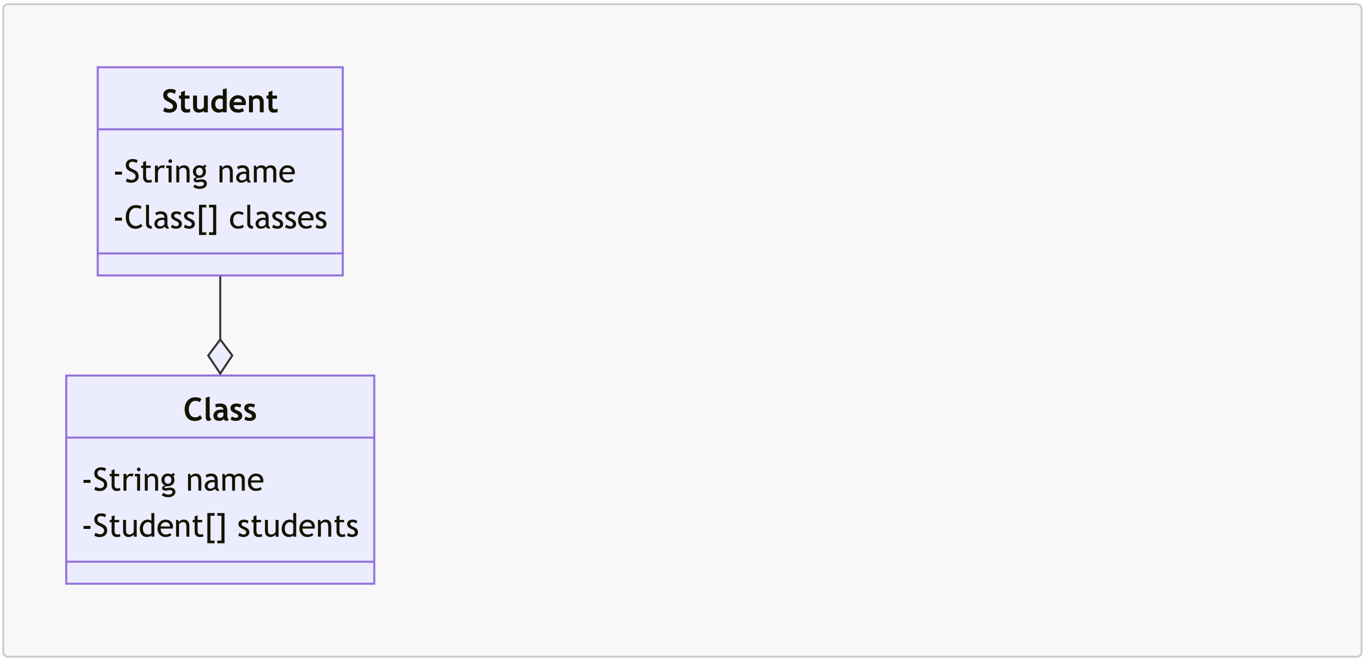


An attribute shared by both entities can only be added to the entity which has multiple instances i.e. the M side.

Many to Many (m:n)

> When multiple instances of entity 1 are linked to multiple instances of entity 2, we have a "many-to-many" relationship. Imagine a scenario where an employee is assigned more than one project.

A student can attend multiple classes and a class can have multiple students.



Schema Design - Case Study

> A schema is a blueprint or plan for a database. It is a collection of logical structures of data, or schema objects, that determine how data is stored and accessed. A schema is a collection of logical structures of data, or schema objects, that determine how data is stored and accessed.

Often, after we build our class diagram, we will need to convert it into a database schema. This is because the database is the most common way to store data. The database schema is the structure of the database.

It is the blueprint of the database. It defines the tables, columns, and relationships between the tables. Roughly the questions answered by the schema are:

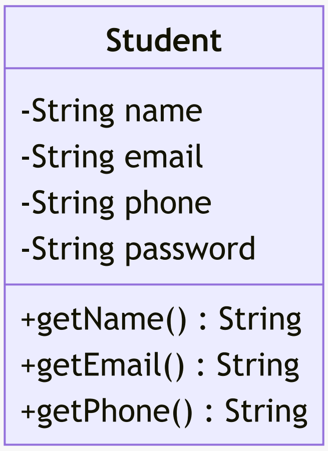
- What data do we need to store corresponding to the request?
- What are different tables we need to create?
- What are the columns in each table?
- What are the keys in each table?

ReScaler - Case Study

Following are the requirements of the ReScaler application:

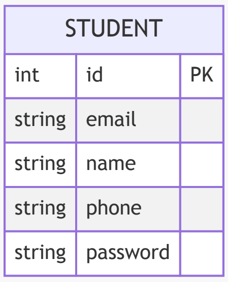
- A student should be able to login with their email address and password.
- Students should be able to view their profile which includes their name, email address and phone number.

Class Diagram



Database Schema

Map the classes as it is to a table in the database. The attributes of the class become the columns of the table.

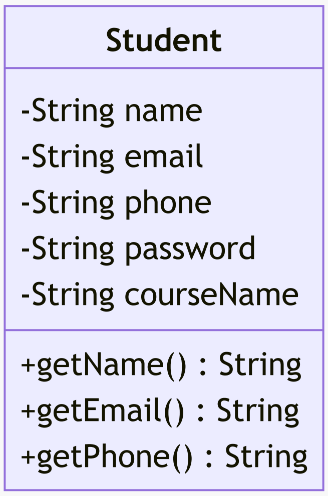


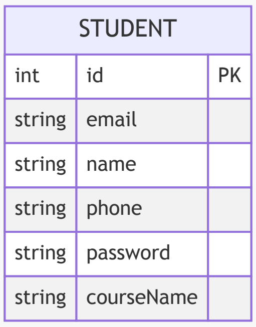
int	id	PK
string	email	
string	name	
string	phone	
string	password	

Adding a new feature

- We will provide multiple courses as a part of the ReScaler application.
- Every student can enroll for a single course.
- Every course has a name.

Course Name as an attribute



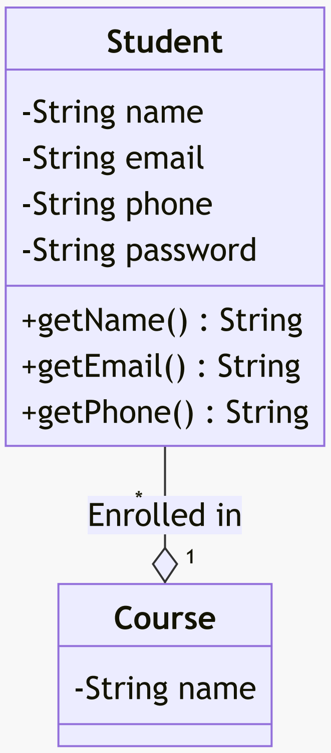


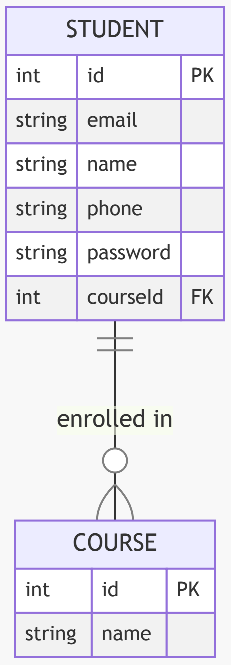
STUDENT		
int	id	PK
string	email	
string	name	
string	phone	
string	password	
string	courseName	

Disadvantages:

- The student can only enroll for a single course.
- Course name is duplicated for every student in the same course. This is a waste of space.
- Updating the course name for all the students is a tedious task.
- Course cannot exist without a student. Our design is subject to database anomalies.

Course as a separate entity



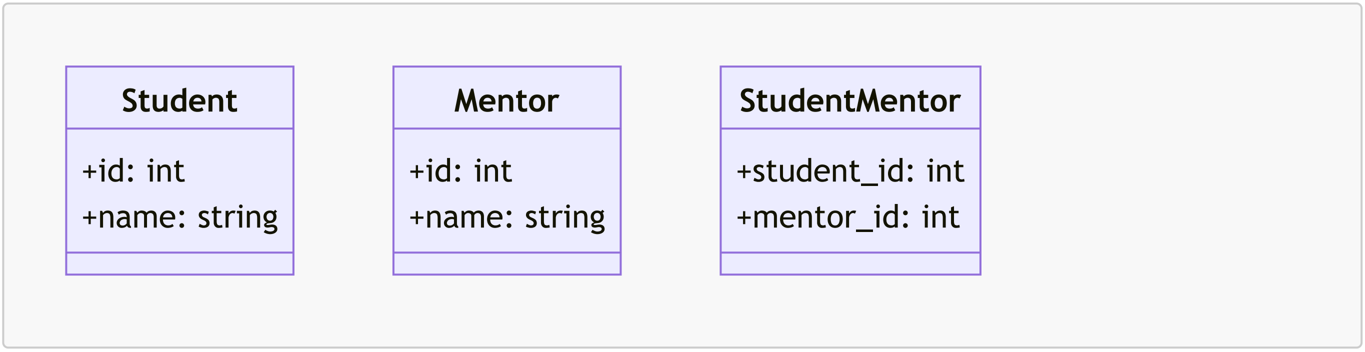


Cardinality - Caveats

Caveat 1: NULL values

Often, when a relationship is not present, we use `NULL` values. For example, a student may not have a mentor. In this case, the `mentor_id` field in the `students` table will be `NULL`. This is a valid value for a foreign key.

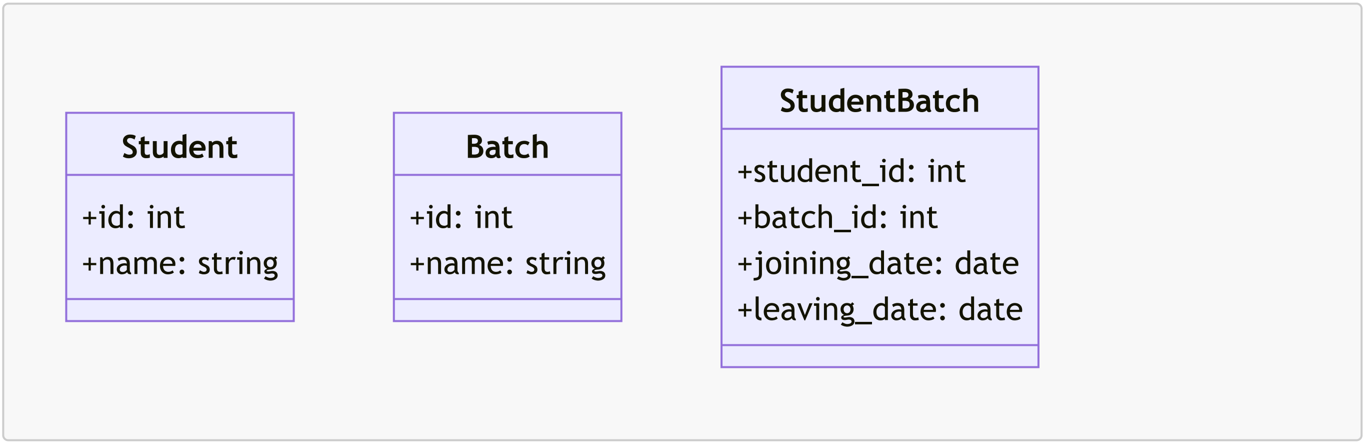
If a table has a lot of NULL values for a foreign key, it is a good idea to create a new mapping table. For example, a student can have a `mentor_id` field with NULL values, we can create a new table `student_mentor`:



Caveat 2: Relations with attributes

Sometimes, a relationship has attributes. For example, a student can have multiple batches. In this case, we can add a `joining_date` and `leaving_date` field to the `student_batch` table.

If the relation attributes are added to the main table, it can get polluted and add to the latency of the table. A better approach is to create a new table with the relation attributes.



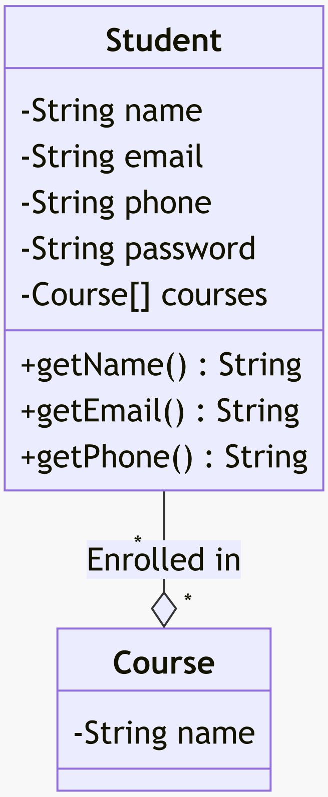
Recap

Cardinality	Normal Relation	Sparse Relation	Relation with Attributes	Example
1:1	Add foreign key on any table	Mapping Table	Mapping Table	Student - Email
1:M	Add foreign key on M side referencing the other	Mapping Table	Mapping Table	Student - Batch
M:N	Mapping Table	Mapping Table	Mapping Table	Student - Class

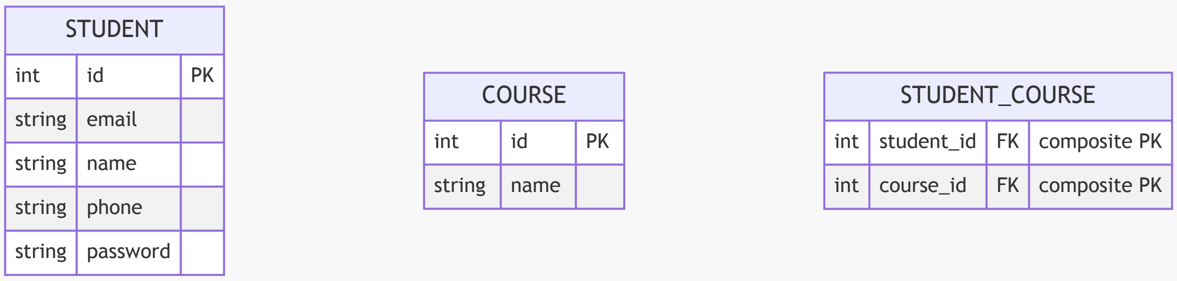
Adding a new feature - II

- A student can now enroll for multiple courses.

In our previous design, we had a `course_id` field in the `students` table. This is a good design for a single course. But, if we want to enroll for multiple courses, the cardinality changes to `M:N`. We need to create a mapping table.



As we have seen in the case of many-to-many relations, we need to create a mapping table.



STUDENT		
int	id	PK
string	email	
string	name	
string	phone	
string	password	

COURSE		
int	id	PK
string	name	

STUDENT_COURSE			
int	student_id	FK	composite PK
int	course_id	FK	composite PK

Recap

1. Identify all the classes in the application.
2. Make a class diagram.
3. Make a table for each class.
4. For all the classes, identify the cardinality of the relationships.
5. Check if any attribute is required to represent the relationship.

Take Home Exercise

Design the class Diagram and database Schema for a system like Netflix with following Use Cases.

- Netflix has users.
- Every user has an email and a password.
- Users can create profiles to have separate independent environments.
- Each profile has a name and a type. Type can be KID or ADULT.
- There are multiple videos on netflix.
- For each video, there will be a title, description and a cast.
- A cast is a list of actors who were a part of the video. For each actor we need to know their name and list of videos they were a part of.
- For every video, for any profile who watched that video, we need to know the status (COMPLETED/ IN PROGRESS).
- For every profile for whom a video is in progress, we want to know their last watch timestamp.