Schema Design: A case study

Key Terms

Schema

> refers to the organization of data as a blueprint of how the database is constructed

> In a relational database, the schema defines the tables, fields, relationships, views, indexes, packages, procedures, functions, queues, triggers, types, sequences, materialized views, synonyms, database links, directories, XML schemas, and other elements

Relational Model

> The relational model (RM) is an approach to managing data using a structure and language consistent with first-order predicate logic, where all data is represented in terms of tuples, grouped into relations

Primary Key

> a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table)

Foreign Key

> A foreign key is a set of attributes in a table that refers to the primary key of another table

Relational DBMS

Using a database in an application leads to extra or boilerplate code. You might see the same piece of code across applications. To reduce this duplication and standardise the code, various data models were proposed.

The most popular being the relational model.

> The relational model (RM) is an approach to managing data using a structure and language consistent with first-order predicate logic, where all data is represented in terms of tuples, grouped into relations

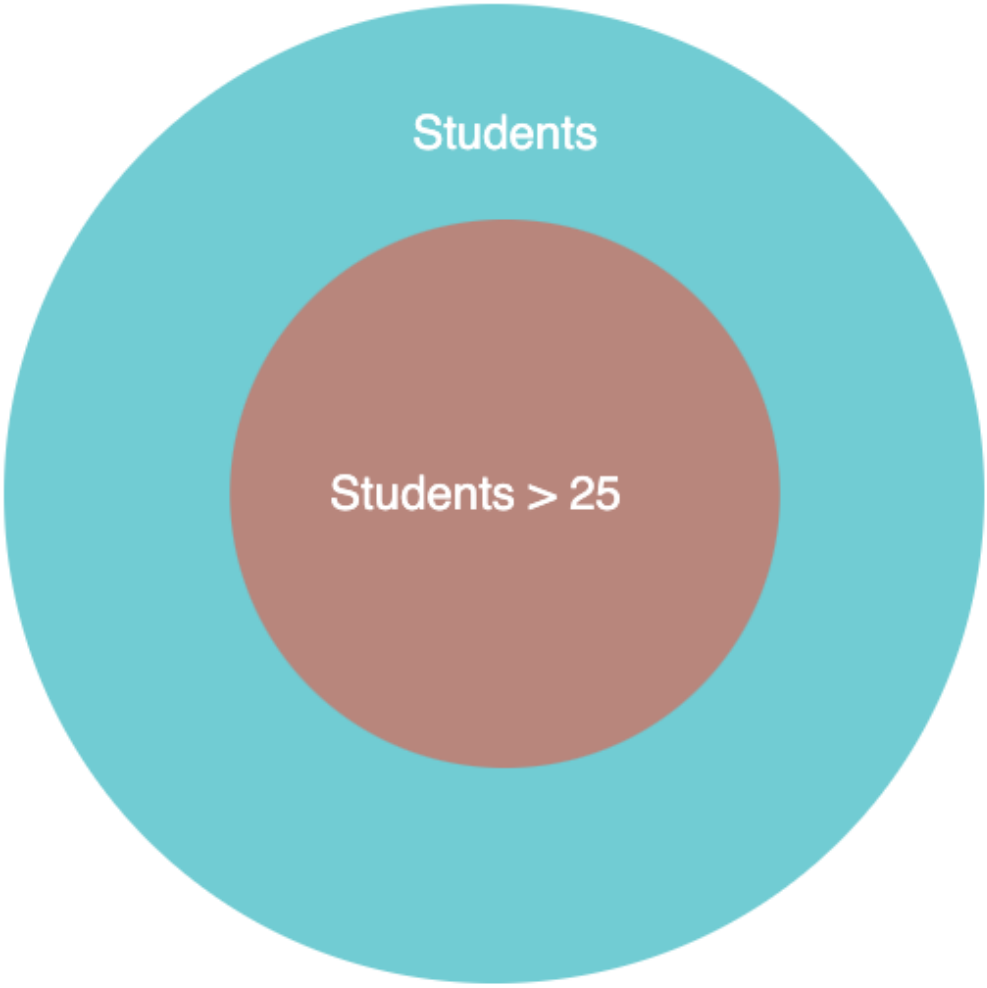
> This relational model has three key points
>
> - Store database in simple data structures (relations).
> - Access data through high-level language.
> - Physical storage left up to implementation.

Relevance to DBMS

SQL Query

```
SELECT * FROM USERS WHERE age > 25;
```

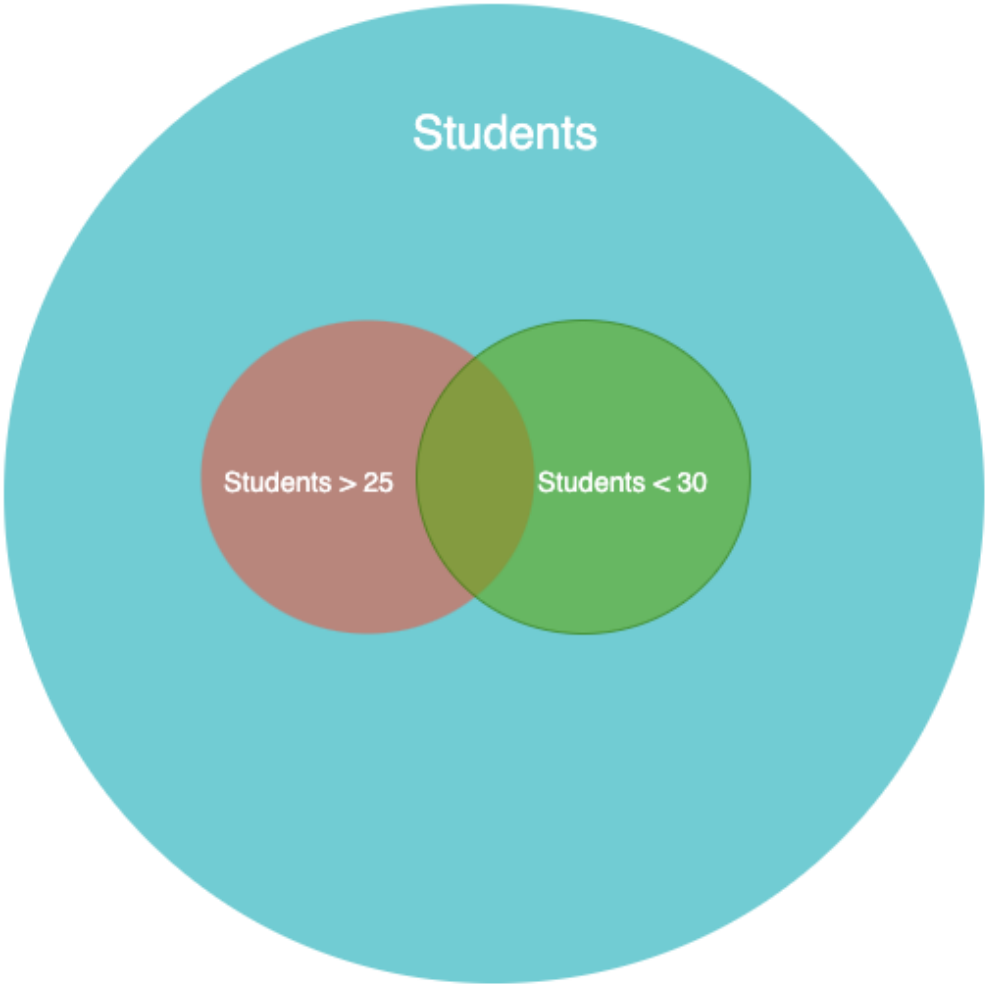
Set operation - Subset



SQL Query

```
SELECT * FROM USERS WHERE age > 25 and age < 30;
```

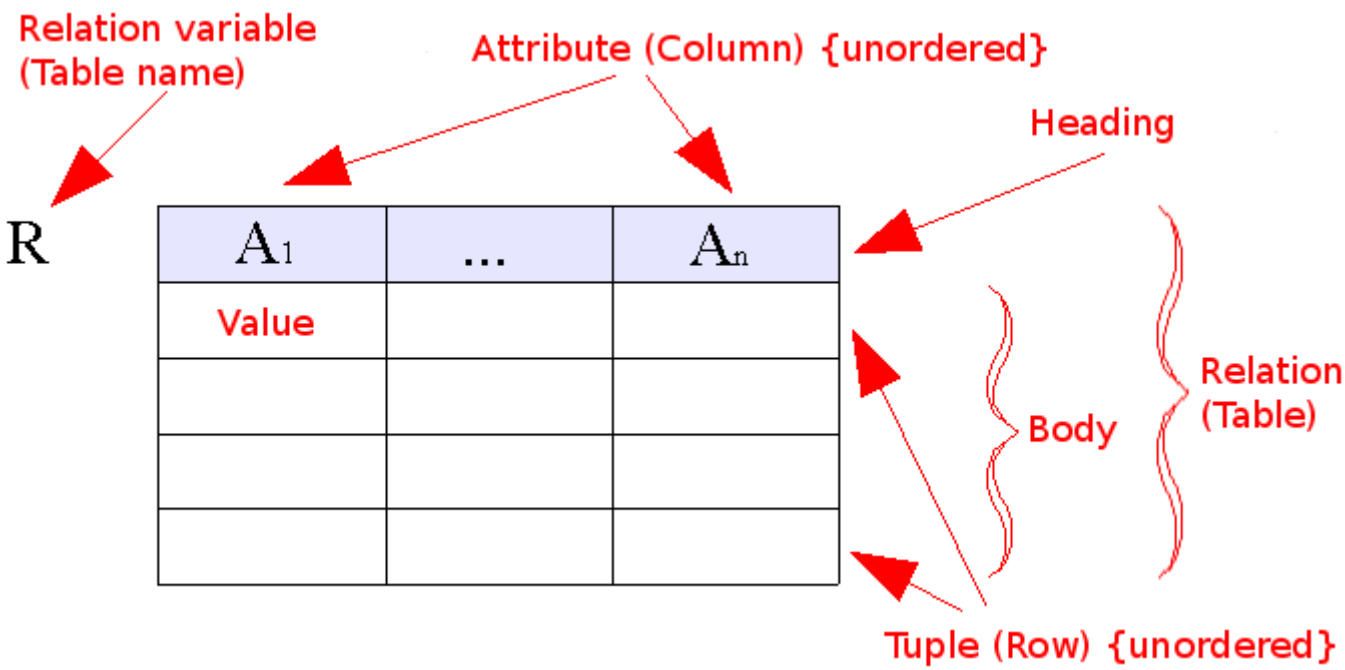
Set operation - Subset



Relational Model

Main features of the relational model are:

- **Relations** - Tables -Represent data as a collection of relations or tables
- **Attributes** - Columns - Each entry in a relation can describe multiple values that are grouped as an attribute
- **Tuples** - Rows - Represent individual data points across multiple attributes



- **Degree** - Number of attributes in a relation
  - > Student (name, age, address, phone, email)
  - Degree - 5
- **Cardinality** - Number of tuples
  - Look at our users file above
  - It has three rows and hence **cardinality is 3**
- **NULL** - For a given tuple, the attribute is undefined.

Properties

- **Uniqueness**
  - Each tuple is unique
  - Each attribute is unique
- **Unordered**
  - Tuples are not ordered
  - Attributes are not ordered
- **Uniform data type** - Every value in a column is of the same data type
- **Atomicity** - Each attribute in each tuple within a relation should consist of a single value and not allow multivalued structures of the kind

---

Keys

Keys are used to

- uniquely identify a tuple in a relation.
- describe relationships between relations.

How can you uniquely identify a student in the students relation?

> Student (name, age, address, phone, email)

- Name
  - A name might not be unique
- Phone or Email
  - Each student will have a unique phone or email address

Super keys

A set of attributes that uniquely identify a tuple in a relation.

For our student relation `Student (name, age, address, phone, email)`, following are some super keys:

- `{id, name}`
- `{id, name, phone}`
- `{id, name, email}`
- `{id, name, email, phone}`
- `{id, name, email, phone, age, address}`
- `{id}`

Candidate keys

A **minimal** set of attributes that uniquely identify a tuple in a relation.

For example `{id, name, email, phone}` is a super key but is it a candidate key?

No, since you can remove `phone` or any other attribute set, but it will still uniquely identify a tuple.

A set of candidate keys for our student relation

- id
- name
- email

Primary Keys

> a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table)

> a primary key is a choice of candidate key (a minimal superkey); any other candidate key is an alternate key.

Each relation can only have one primary key

Composite Keys

Sometime you want to use multiple attributes to uniquely identify a tuple. For example, you might want to use a combination of name and phone number to uniquely identify a student.

Composite keys are used in mapping tables. For instance, you might want to have a relation for student feedback. You could use `student_id` and `batch_id` for uniquely identifying a tuple.

Foreign Keys

> A Foreign Key is a database key that is used to link two tables together A foreign key is a set of attributes in a table that refers to the primary key of another table

Imagine adding a batch to a student. You could add all the column for a batch in the student relation.

Name	Email	Phone	Age	Address	Batch Name	Batch	Start Date	Type

Problem with this solution

- Duplication - All the columns from the batch relation are duplicated
- Integrity - What if the original relation is duplicated?

Can we just reference the original relation?

Name	Email	Phone	Age	Address	Batch Id

Some queries to get you started

Create the students relation

```
CREATE TABLE `students` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `age` int DEFAULT NULL,
  `phone` int DEFAULT NULL,
  `email` varchar(255) NOT NULL,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`);
)
```

Create the mentors relation

```
CREATE TABLE `mentors` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `age` int DEFAULT NULL,
  `phone` int DEFAULT NULL,
  `address` varchar(255) DEFAULT NULL,
  `email` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
);
```

Create the batches relation

```sql
CREATE TABLE `batches` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `start_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `type` varchar(255) NOT NULL,
  `mentor_id` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `mentor_id` (`mentor_id`),
  CONSTRAINT `batches_ibfk_1` FOREIGN KEY (`mentor_id`) REFERENCES
`mentors` (`id`)
);
```

Schema design

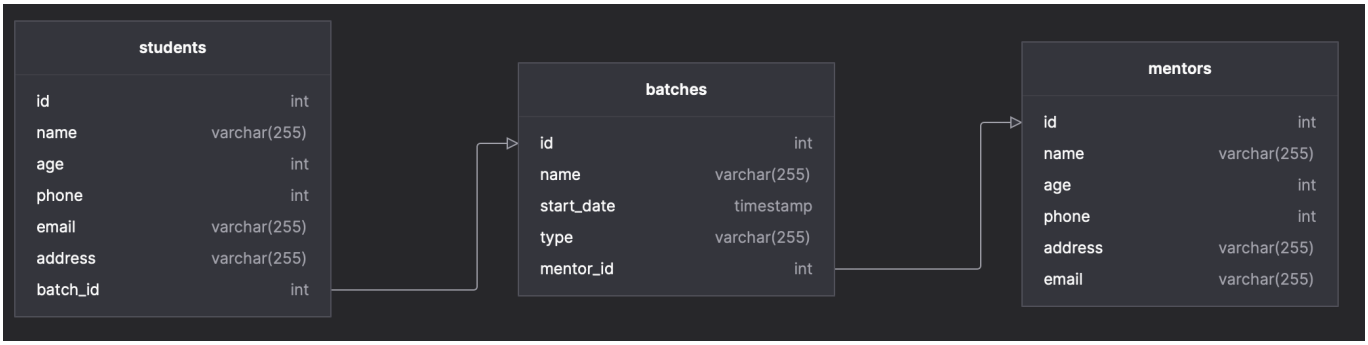
A schema is a blueprint of a database. It is created before you actually construct the database so that the schema design can be reviewed. Schema diagrams are also a great way to document the database structure in one place.

Remember our student's database from the previous lesson? We had the three following tables

- `students` (id, name, age, address, phone, email, batch ID)
- `mentors` (id, name, age, address, phone, email)
- `batches` (id, name, mentor, start date, type, mentor ID)

So each table has `ID` as primary key. The `students` table has a `batch ID` field that references the `batches` table and the `batches` table has a `mentor ID` field that references the `mentors` table. These are examples of foreign keys. These are some the items that are present in a schema. A schema will also contain indexes, constraints, and other items that are present in a table.

Following is a schema diagram for the above database. Note that the primary key is not highlighted here, which ideally should be.



> **Note**
>
> Try it yourself.
> Go to this website and import this diagram.
> Try adding a new column or even a new table.

Case study: Requirements

- There are several batches at Scaler. Each batch has an ID, name, current instructor.
- Each batch has multiple classes. Each class has an ID, name, instructor
- Every Student has a name, ID, grad year, university, email, phone number, ctc, current batch.
- Every student also has a student buddy.
- A student may have been moved from one batch to another due to pausing the course. So we need to know the entry date and leaving date of a student on every batch they were a part of..
- Every student has a mentor. Every mentor has a name, dob.

You can also find these here.

Initial design

The first step of designing a schema is to identify the entities. This can be done by identifying the nouns in the requirements. For example, take the first requirement.

> There are several batches at Scaler. Each batch has an ID, name, current instructor.

We can identify the following entities:

- Batches
- Instructor

Running through the requirements, we can identify the following entities:

- Batches
- Instructor
- Students
- Classes
- Mentor

The next step would be to identify the attributes of each entity. For example, the `Batches` entity has the following attributes:

- ID
- Name
- Current Instructor
- Classes

The initial set of tables would look like this:

```
classDiagram
    class Batches{
        + ID
        + Name
        + Current Instructor
        + Classes
    }
    class Instructor{
        + ID
        + Name
        + DOB
```

```
    }
    class Students{
        + ID
        + Name
        + Grad Year
        + University
        + Email
        + Phone Number
        + CTC
        + Current Batch
        + Student Buddy
        + Mentor
        + Previous Batches
    }
    class Classes{
        + ID
        + Name
        + Instructor
    }
    class Mentor{
        + ID
        + Name
        + DOB
    }
```

The initial design has the following issues:

- Foreign keys are not present
- Attributes are not atomic
- There is no way to know the entry and leaving date of a student on a batch

The next step would be to identify the relationships between the entities to add the foreign keys. To identify relationships, we need to find the cardinality of the relations.

Cardinality

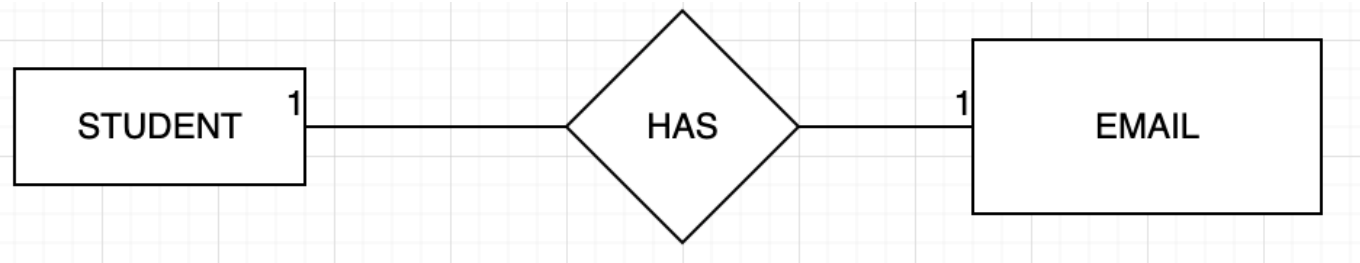
> Cardinality is the maximum times an entity can relate to an instance with another entity or entity set.

> 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.

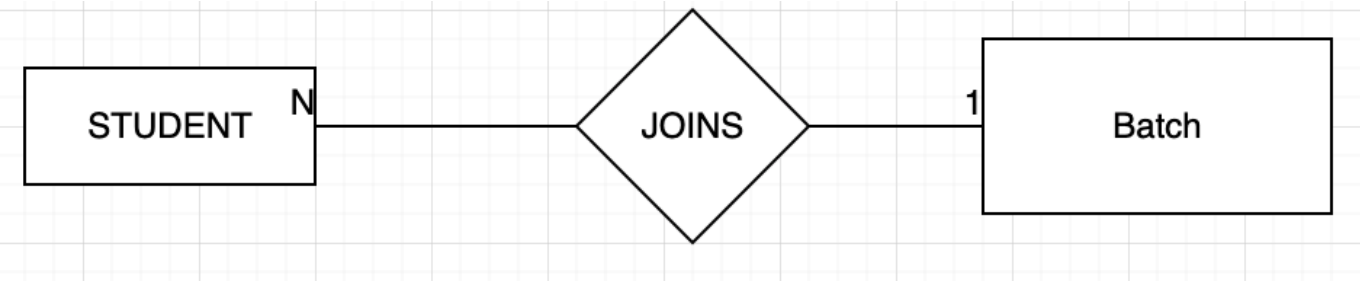


An attribute shared by both entities can be added to either of the entities.

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.

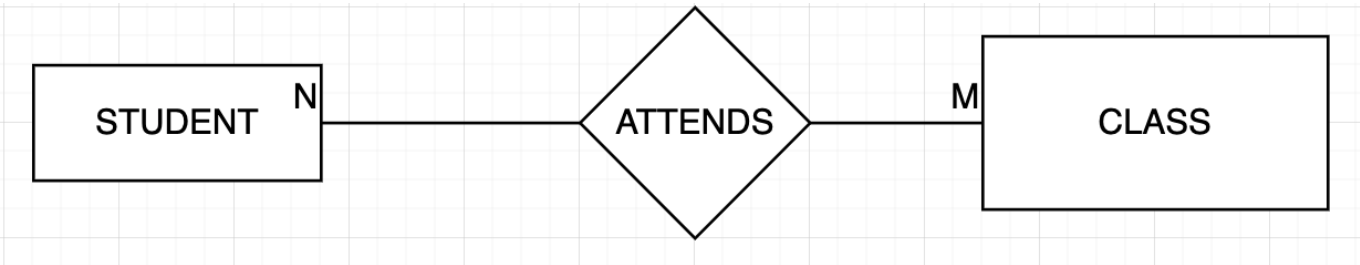


An attribute shared by both entities can only be added to the entity which has multiple instances i.e. the M side.

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.



An attribute shared by both entities has to be added to the relationship.

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`:

```
classDiagram
    class Student{
        +id: int
        +name: string
    }
    class Mentor{
        +id: int
        +name: string
    }
    class StudentMentor{
        +student_id: int
        +mentor_id: int
    }
```

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.

```
classDiagram
    class Student{
        +id: int
        +name: string
    }
    class Batch{
        +id: int
        +name: string
    }
    class StudentBatch{
        +student_id: int
        +batch_id: int
        +joining_date: date
        +leaving_date: date
    }
```

Recap

Cardinality	Normal Relation	Sparse Relation	Relation with Attributes	Example
1:1	Add foreign key on any table	Mapping Table	Mapping Table	Student - Email

Cardinality	Normal Relation	Sparse Relation	Relation with Attributes	Example
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

Final Design

Now that we know about cardinality, we can go ahead and identify the various relationships between the entities.

- A batch and an instructor have a Many to One relationship. An instructor can teach multiple batches, but a batch can only have one instructor. So we add a foreign key `instructor_id` to the `batches` table.

```
classDiagram
    class Batches{
        + ID
        + Name
        + instructor_id
        + Classes
    }
    class Instructor{
        + ID
        + Name
        + DOB
    }
    Batches "M" -- "1" Instructor
```

- A batch can have multiple classes and a class can be a part of multiple batches i.e. Many to Many relationship. So we create a mapping table `batch_classes` with the attributes `batch_id` and `class_id`.

```
classDiagram
    class Batches{
        + ID
        + Name
        + instructor_id
    }
    class Classes{
        + ID
        + Name
        + Instructor
    }
    class BatchClass{
        + batch_id
```

```
        + class_id
    }
```

- A class can have one instructor and an instructor can teach multiple classes i.e. Many to One relationship. So we add a foreign key `instructor_id` to the `classes` table.

```
classDiagram
    class Classes{
        + ID
        + Name
        + instructor_id
    }
    class Instructor{
        + ID
        + Name
        + DOB
    }
    Classes "M" -- "1" Instructor
```

- A student can have one mentor and a mentor can have multiple students i.e. Many to One relationship. So we add a foreign key `mentor_id` to the `students` table.

```
classDiagram
    class Students{
        + ID
        + Name
        + Grad Year
        + University
        + Email
        + Phone Number
        + CTC
        + Current Batch
        + Student Buddy
        + mentor_id
        + Previous Batches
    }
    class Mentor{
        + ID
        + Name
        + DOB
    }
    Students "M" -- "1" Mentor
```

- A student can be a part of only one batch at a time, but a batch can have multiple students i.e. Many to One relationship. So we add a foreign key `batch_id` to the `students` table.

```
classDiagram
    class Students{
        + ID
        + Name
        + Grad Year
        + University
        + Email
        + Phone Number
        + CTC
        + Student Buddy
        + mentor_id
        + Previous Batches
        + batch_id
    }
    class Batches{
        + ID
        + Name
        + instructor_id
    }
    Students "M" -- "1" Batches
```

- A student can have multiple previous batches and a batch can have multiple students i.e. Many to Many relationship. So we create a mapping table `student_batch` with the attributes `student_id` and `batch_id`. Also, we need to store the `joining_date` and `leaving_date` for each student-batch relationship.

```
classDiagram
    class Students{
        + ID
        + Name
        + Grad Year
        + University
        + Email
        + Phone Number
        + CTC
        + Student Buddy
        + mentor_id
        + batch_id
    }
    class Batches{
        + ID
        + Name
        + instructor_id
    }
    class StudentBatch{
        + student_id
        + batch_id
        + joining_date
        + leaving_date
    }
```

The complete final design is as follows:

```
classDiagram
    class Students{
        + id
        + name
        + grad_year
        + university
        + email
        + phone_number
        + ctc
        + student_buddy_id
        + mentor_id
        + batch_id
    }
    class Mentor{
        + id
        + name
        + dob
    }

    class Batches{
        + id
        + name
        + instructor_id
    }

    class Classes{
        + id
        + name
        + instructor_id
    }

    class Instructor{
        + id
        + name
        + dob
    }

    class StudentBatch{
        + student_id
        + batch_id
        + joining_date
        + leaving_date
    }

    class BatchClass{
        + batch_id
        + class_id
    }

    Students "M" -- "1" Mentor
    Students "M" -- "1" Batches
```

```
    Batches "M" -- "1" Instructor
    Classes "M" -- "1" Instructor
```